ERDC/EL TR-01-32

US Army Corps
of Engineers®
Engineer Research and
Development Center

In-House Laboratory Independent Research Program

Computed-Tomography Imaging SpectroPolarimeter (CTISP) – A Passive Optical Sensor

Volume 2, Appendix B

Hollis H. (Jay) Bennett, Jr., Ricky A. Goodson, and
John O. Curtis

September 2001

Environmental Laboratory

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

PRINTED ON RECYCLED PAPER

In-House Laboratory Independent Research Program

ERDC/EL TR-01-32
September 2001

Computed-Tomography Imaging SpectroPolarimeter (CTISP) – A Passive Optical Sensor

Volume 2, Appendix B

By Hollis H. (Jay) Bennett, Jr., Ricky A. Goodson, John O. Curtis

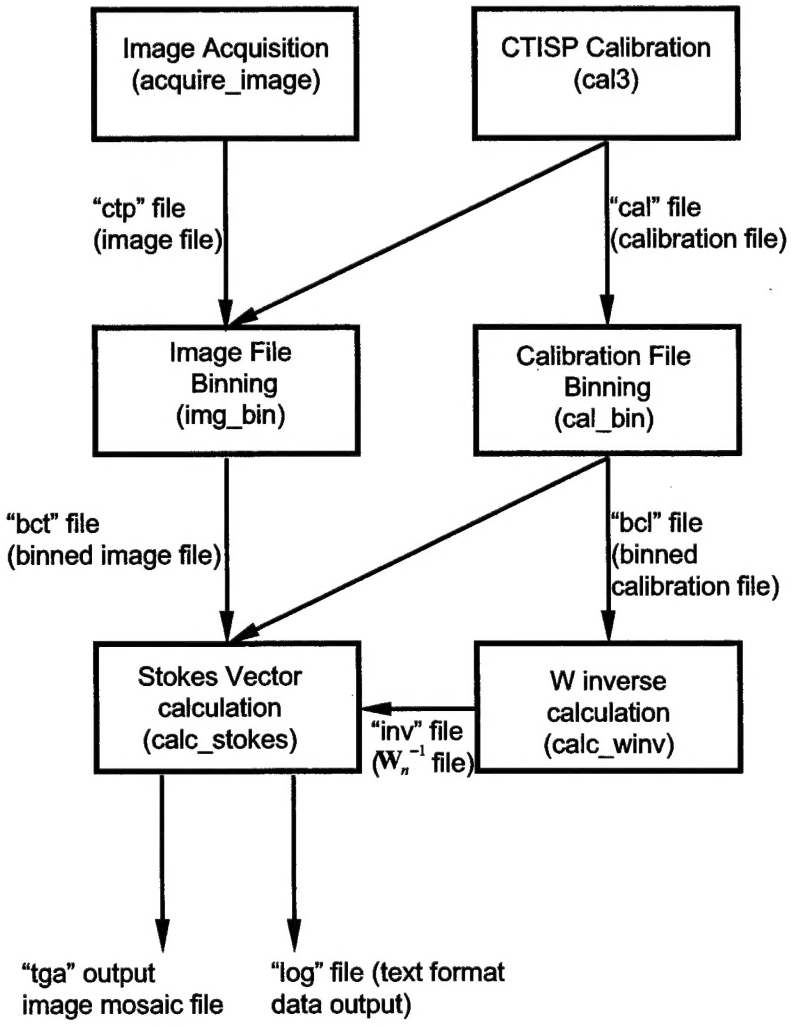
Environmental Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final report

Prepared for Assistant Secretary of the Army
Washington, DC 20315

Appendix B
Code Listings

The relationships among the computer programs that were developed to acquire and process data with the Computed-Tomograph Imaging SpectroPolarimeter (CTISP) are shown in the following flow diagram. The three programs on the right side of the diagram involve the calibration facility and are only executed when a new calibration matrix is needed. The three programs on the left side of the diagram are the programs that are executed for each scene that is acquired. Code listings for these six programs are given on the following pages. In addition, a code listing for one other program, called cal-utility, is also provided. This program is not used during standard operation of the system, but is used to facilitate maintenance of the calibration facility.



Cal3

Files: cal3.cpp
cal3Dlg.cpp
camera.cpp
monochrometer.cpp
pol_states.cpp
xyposition.cpp
sioerror.c
cal3.h
cal3Dlg.h
camera.h
param.h
pol_states.h
xyposition.h

cal3.cpp

```
// cal3.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "cal3.h"
#include "cal3Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCal3App

BEGIN_MESSAGE_MAP(CCal3App, CWinApp)
    //{{AFX_MSG_MAP(CCal3App)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCal3App construction

CCal3App::CCal3App()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```

```
/////////////////////////////////////////////////////////////////////////////
// The one and only CCal3App object

CCal3App theApp;

/////////////////////////////////////////////////////////////////////////////
// CCal3App initialization

BOOL CCal3App::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();         ..... // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();// Call this when linking to MFC statically
#endif

    CCal3Dlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

cal3Dlg.cpp

```
// cal3Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "cal3.h"
#include "cal3Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCal3Dlg dialog

CCal3Dlg::CCal3Dlg(CWnd* pParent /*=NULL*/)
    : CDialog(CCal3Dlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CCal3Dlg)
    m_winc = 20;
    m_wstart = 440;
    m_wsteps = 16;
    m_xstart = 495;
    m_ystart = 516;
    m_pstate = _T("");
    m_wave = 0;
    m_imax = 0;
    m_imean = 0;
    m_imin = 0;
    m_minusx = 0;
    m_minusy = 0;
    m_plusx = 0;
    m_plusy = 0;
    m_Status_Edit = _T("");
    m_xpos = 0.0f;
    m_ypos = 0.0f;
    m_pol = _T("");
    m_iexp = 0.012f;
    m_outfile = _T("");
    m_cur_exp = 0;
    m_tot_exp = 0;
    m_cur_wp = 0.0f;
    pstates = 0;
    m_exp = 0.0f;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCal3Dlg::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCal3Dlg)
    DDX_Control(pDX, IDC_POL_COMBO, m_pol_con);
    DDX_Control(pDX, IDC_STATUS_BOX, m_Status_Con);
    DDX_Text(pDX, IDC_WINC_BOX, m_winc);
    DDX_Text(pDX, IDC_WSTART_BOX, m_wstart);
    DDV_MinMaxUInt(pDX, m_wstart, 350, 800);
    DDX_Text(pDX, IDC_WSTEPS_BOX, m_wsteps);
    DDX_Text(pDX, IDC_XSTART_BOX, m_xstart);
    DDV_MinMaxUInt(pDX, m_xstart, 420, 540);
    DDX_Text(pDX, IDC_YSTART_BOX, m_ystart);
    DDV_MinMaxUInt(pDX, m_ystart, 456, 574);
    DDX_Text(pDX, IDC_PSTATE_BOX, m_pstate);
    DDX_Text(pDX, IDC_WAVE_BOX, m_wave);
    DDX_Text(pDX, IDC_IMAX_BOX, m_imax);
    DDX_Text(pDX, IDC_IMEAN_BOX, m_imean);
    DDX_Text(pDX, IDC_IMIN_BOX, m_imin);
    DDX_Text(pDX, IDC_MINUSX_BOX, m_minusx);
    DDX_Text(pDX, IDC_MINUSY_BOX, m_minusy);
    DDX_Text(pDX, IDC_PLUSX_BOX, m_plusx);
    DDX_Text(pDX, IDC_PLUSY_BOX, m_plusy);
    DDX_Text(pDX, IDC_STATUS_BOX, m_Status_Edit);
    DDX_Text(pDX, IDC_XPOS_BOX, m_xpos);
    DDX_Text(pDX, IDC_YPOS_BOX, m_ypos);
    DDX_CBString(pDX, IDC_POL_COMBO, m_pol);
    DDX_Text(pDX, IDC_IEXP_BOX, m_iexp);
    DDX_Text(pDX, IDC_OUTFILE_BROWSE_BOX, m_outfile);
    DDX_Text(pDX, IDC_CURRENT_EXP_BOX, m_cur_exp);
    DDX_Text(pDX, IDC_TOT_EXP_BOX, m_tot_exp);
    DDX_Text(pDX, IDC_WP_BOX, m_cur_wp);
    DDX_Text(pDX, IDC_EXP_BOX, m_exp);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCal3Dlg, CDialog)
    //{{AFX_MSG_MAP(CCal3Dlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_CAL_BUTTON, OnCalButton)
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
    ON_BN_CLICKED(IDC_ONE_POL_RADIO, OnOnePolRadio)
    ON_BN_CLICKED(IDC_ALL_POL_RADIO, OnAllPolRadio)
    ON_BN_CLICKED(IDC_OUTFILE_BROWSE_BUTTON,
OnOutfileBrowseButton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
```

```
// CCal3Dlg message handlers

BOOL CCal3Dlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);..... // Set big icon
    SetIcon(m_hIcon, FALSE);.....// Set small icon

    // TODO: Add extra initialization here

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CCal3Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
```

```
// this is automatically done for you by the framework.

void CCal3Dlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CCal3Dlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CCal3Dlg::OnCalButton()
{
    int iResult, i, m, n, k;
    int xguess, yguess, wave;
    FILE *outfile;
    unsigned short buffer[CCD_WIDTH*CCD_HEIGHT];
    BOOL bUseHighGain=1;
    BOOL bUseROI=1;
    int nMessageMode=NO_MESSAGES;
    struct stats_st image_stats;
    int nonzero;
    unsigned short *buff;
    float exp_time, exp[100];
    int pol_pos[POL_STATES+1];
    char mess[200];
```

```
    char *pol_st[POL_STATES+1]={"Circular", "Vertical", "45 Degrees",
            "Horizontal", "None"};
    int pol, pol_order[POL_STATES+1];
    double wave_voltage[100];
    int xp, xm, yp, ym;
    char fn[200];
    int iflt, init_filter, last_filter, nfilters;

    // Get parameters
    UpdateData (TRUE);

    // Open output file
    strcpy (fn, (LPCTSTR)m_outfile);
    if (fn[0] == '\0') {
        MessageBox ("No Output File Selected", "Cal", MB_OK);
        return;
    }
    if( (outfile = fopen(fn,"w" )) == NULL ) {
        MessageBox("Error opening calibration file", "Cal", MB_OK);
        return;
    }

    // Make sure that the Frame width defined by the ROI is even.
    // An odd value will result in an image with line to line
    // horizontal shifts due to a bug in the camera software.
    if ((FRAME_WIDTH % 2) != 0) {
        MessageBox ("Frame width must be even. Adjust ROI parameters.",
                "Calibrate", MB_OK);
        OnExitButton ();
    }

    // Determine filters to use
    pol = m_pol_con.GetCurSel ();
    if (pol == 5) {
        init_filter = 1;
        last_filter = 4;
        nfilters = 4;
    }
    else {
        init_filter = pol;
        last_filter = pol;
        nfilters = 1;
    }

    // Determine order for polarization states
    if (pstates == 0) {
        MessageBox ("Polarization states must be selected", "Calibrate",
MB_OK);
        return;
    }
```

```
else if (pstates == 1)
    pol_order[0] = pol;
else {
    for (i=0; i<pstates; i++)
        pol_order[i] = i;
    pol = 1;
}

// Calculate positions for polarization states
pol_pos[0] = 0;
pol_pos[1] = 0;
pol_pos[2] = (int)(45.0 * ((float)STEPS_PER_REV/360.0) + .5);
pol_pos[3] = (int)(90.0 * ((float)STEPS_PER_REV/360.0) + .5);
pol_pos[4] = 0;

// Calculate number of exposures and display
m_tot_exp = m_wsteps*pstates*nfilters;
m_cur_exp = 0;
UpdateData (FALSE);
update_status_scroll ();

// Set up monochrometer
iResult = monochrometer_setup ();
if (iResult!=0) {
    MessageBox ("Error in monochrometer setup", "Calibrate", MB_OK);
    OnExitButton ();
}
m_Status_Edit += "Monochrometer set up complete\r\n";
UpdateData (FALSE);
update_status_scroll ();

// Set up camera
iResult = camera_setup(bUseHighGain, bUseROI, nMessageMode, m_iexp);
if (iResult!=0) {
    sprintf(mess, "Error: camera_setup returned %i\n",iResult);
    MessageBox (mess, "Calibrate", MB_OK);
    OnExitButton ();
}
m_Status_Edit += "Camera set up complete\r\n";
UpdateData (FALSE);
update_status_scroll ();

// Configure ports for wave plate, photodetector, polarizer, and pol filter
iResult = waveplate_setup ();
if (iResult!=0) {
    sprintf(mess, "Error: waveplate_setup returned %i\n",iResult);
    MessageBox (mess, "Calibrate", MB_OK);
    OnExitButton ();
}
iResult = photodetector_setup ();
```

```
    if (iResult!=0) {
        sprintf(mess, "Error: photodetector_setup returned %i\n",iResult);
        MessageBox (mess, "Calibrate", MB_OK);
        OnExitButton ();
    }
    polarizer_setup ();
    polarizer_reset ();
    pol_filter_setup ();
    m_Status_Edit += "waveplate, photodetector, polarizer, and pol filter set up
complete\r\n";
    UpdateData (FALSE);
    update_status_scroll ();

    // calibrate polarizer movement
    if (pstates == 4 || ((pol > 0) && (pol < 4))) {
        m_Status_Edit += "Calibrating polarizer position\r\n";
        UpdateData (FALSE);
        update_status_scroll ();
        polarizer_reset ();
    }

    // Calibrate circular polarization
    if (pstates == 4 || pol == 0) {
        for (m=0; m<m_wsteps; m++) {
            wave = m_wstart+m*m_winc;
            wave_voltage[m] = circular_pol (wave);
        }
    }

    // Make sure mirror is down
    Flip_Mirror (down);

    // calculate exposure time for middle wavelength
    set_wp_voltage (5.0);
    wave = m_wstart+(m_wsteps/2)*m_winc;
    if (pol==0)
        set_wp_voltage (wave_voltage[m]);
    GoToWavelength (wave);
    m_wave = wave;
    UpdateData (FALSE);
    update_status_scroll ();
    exp_time = set_exposure (buffer, m_iexp);

    // Set exposure for middle wavelength
    iResult=camera_setup (bUseHighGain, bUseROI, nMessageMode,
exp_time);
    if (iResult!=0) {
        sprintf(mess, "Error: camera_setup returned %i\n",iResult);
        MessageBox (mess, "Calibrate", MB_OK);
        OnExitButton ();
```

```
}
sprintf (mess, "Exposure time set to %f\r\n", exp_time);
m_Status_Edit += mess;
UpdateData (FALSE);
update_status_scroll ();

// Determine initial position of fiber
//
iResult=pvAcquireFrame(BOARDNUM,buffer);
if (iResult!=0) {
        sprintf (mess, "ERROR pvacquireframe returned %i\n",iResult);
        MessageBox (mess, "Calibrate", MB_OK);
}
stats (buffer, &image_stats);
pos = find_center   (buffer, image_stats.mean*thresh);
m_xstart = (int)(pos->x + 0.5);
m_ystart = (int)(pos->y + 0.5);
xguess = m_xstart;
yguess = m_ystart;
m_Status_Edit += "Fiber position calculated\r\n";
UpdateData (FALSE);
update_status_scroll ();

// Calculate camera exposure time at each wavelength
//
set_wp_voltage (5.0);
for (m=0; m<m_wsteps; m++) {
    wave = m_wstart+m*m_winc;
    if (pol==0)
        set_wp_voltage (wave_voltage[m]);
    GoToWavelength (wave);
    m_wave = wave;
    UpdateData (FALSE);
    update_status_scroll ();
    exp[m] = set_exposure (buffer, m_iexp);
    exp[m] = set_exposure (buffer, exp[m]);
    sprintf (mess, "Exposure time for %dnm is %f\r\n", wave, exp[m]);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();
}

// Write header info
fprintf (outfile, "%d %d\n", FRAME_WIDTH, FRAME_HEIGHT);
fprintf (outfile, "1 1\n");
fprintf (outfile, "%d 1 1\n", m_xstart);
fprintf (outfile, "%d 1 1\n", m_ystart);
fprintf (outfile, "%d\n", nfilters);
fprintf (outfile, "%d %d %d\n", m_wstart, m_winc, m_wsteps);
fprintf (outfile, "%d\n", pstates);
```

```
fprintf (outfile, "%d\n", m_wsteps*pstates*nfilters);

// Acquire calibration images
//
for (iflt=init_filter; iflt<=last_filter; iflt++) {
   rotate_pol_filter (iflt*STEPS_PER_FILTER+POL_FILTER_OFFSET);
   for (m=0; m<m_wsteps; m++) {

      // Set wavelength and exposure time
      //
      wave = m_wstart+m*m_winc;
      GoToWavelength (wave);
      m_wave = wave;
      m_exp = exp[m];
      exp_time = exp[m];
      iResult=camera_setup (bUseHighGain, bUseROI, nMessageMode,
                            exp_time);
      if (iResult!=0)
         printf("ERROR camera_setup returned %i\n",iResult);

      // rotate through polarization states
      for (n=0; n<pstates; n++) {
         rotate_polarizer (pol_pos[pol_order[n]], ABS_MODE);
         if (pol_order[n]==0)
            set_wp_voltage (wave_voltage[m]);
         else
            set_wp_voltage (5.0);
         m_pstate = pol_st[pol_order[n]];
         Sleep (1000);

         // Acquire image
         //
         m_cur_exp++;
         UpdateData (FALSE);
         update_status_scroll ();
         iResult=pvAcquireFrame(BOARDNUM,buffer);
         if (iResult!=0) {
            sprintf (mess, "ERROR pvacquireframe returned %i\n",iResult);
            MessageBox (mess, "Calibrate", MB_OK);
         }
         /* These lines are needed for radiometric calibration
         iResult = read_photodetector ();
         fprintf (outfile, "%d %d %d\n", pol_order[n], wave, iResult);
         m_imean = iResult;
         UpdateData (FALSE);
         update_status_scroll ();
         */
         // Calculate stats and zero pixels below threshold
         //
         stats (buffer, &image_stats);
```

```
                nonzero = zero (buffer, 3.0, xguess, yguess);.....
                fprintf (outfile, "%d %d %d %d %d %d %f\n", xguess, yguess,
                    iflt, pol_order[n], wave, nonzero, exp_time);
                buff = buffer;
                for (k=0; k<(FRAME_WIDTH*FRAME_HEIGHT); k++) {
                    if ((*buff) > 0)
                        fprintf (outfile, "%d %d\n", k, *buff);
                    buff++;
                }
            }
        }
    }

    // Move polarizer and filter wheel to starting positions
    rotate_pol_filter (0);
    rotate_polarizer (0, ABS_MODE);

    fclose (outfile);
    com_close (POL_PORT);
    com_close (POL_FILTER_PORT);
    MessageBox ("Done", "Calibrate", MB_OK);
    return;

}


float CCal3Dlg::set_exposure (unsigned short *buffer, float exp_time)
{
    int iResult;
    struct stats_st image_stats;
    char mess[200];
    BOOL bUseHighGain=1;
    BOOL bUseROI=1;
    int nMessageMode=NO_MESSAGES;

    // Set wavelength to initial setting
    //
    iResult=camera_setup (bUseHighGain, bUseROI, nMessageMode,
exp_time);
    if (iResult!=0) {
        sprintf(mess, "Error: camera_setup returned %i\n",iResult);
        MessageBox (mess, "Calibrate", MB_OK);
        OnExitButton ();
    }
    sprintf (mess, "Exposure time set to %f\r\n", exp_time);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();

    // acquire frame
```

```
    //
    iResult=pvAcquireFrame(BOARDNUM,buffer);
    if (iResult!=0) {
        sprintf (mess, "ERROR pvacquireframe returned %i\n",iResult);
        MessageBox (mess, "Calibrate", MB_OK);
    }

    // Calculate correct exposure for this wavelength
    //
    stats (buffer, &image_stats);
    if (image_stats.max == MAX_INTENSITY) {
        sprintf (mess, "image saturated, exposure time = %f", m_iexp);
        iResult = MessageBox (mess, "calibrate", MB_OKCANCEL);
        if (iResult == IDCANCEL)
            OnExitButton ();
    }
    m_exp = (float)(exp_time * ((intensity_thresh*MAX_INTENSITY) /
                (image_stats.max-image_stats.mean)));
    m_imin = image_stats.min;
    m_imax = image_stats.max;
    m_imean = image_stats.mean;
    sprintf (mess, "Calculated exposure time is %f\r\n", m_exp);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();
    return (m_exp);
}


void CCal3Dlg::OnExitButton()
{
    DestroyWindow ();
    exit (0);
}


void CCal3Dlg::update_status_scroll ()
{
    int minscr, maxscr;


    m_Status_Con.GetScrollRange (SB_VERT, &minscr, &maxscr);
    if (maxscr > 11)
        m_Status_Con.LineScroll (maxscr-11, 0);
    UpdateWindow ();
}


// zero - zeroes all pixel values less than a threshold
//
```

```
int CCal3Dlg:: zero (unsigned short *buffer, float threshold, int xguess,
                     int yguess)
{
    unsigned short *buff;
    int i, j, k;
    int num;
    double sum, avg, sd;
    char mess[100];
    int val;

    xguess = xguess - ROI_LEFT;
    yguess = yguess - ROI_TOP;
    sum = 0.0;
    for (i=0; i<30; i++)
        for (j=0; j<100; j++)
            sum += *(buffer+i*FRAME_WIDTH+j);
    avg = sum / 3000.0;
    sum = 0;
    for (i=0; i<30; i++)
        for (j=0; j<100; j++) {
            val = avg - *(buffer+i*FRAME_WIDTH+j);
            sum += val * val;
        }
    sd = sqrt(sum/2999.0);
    num = 0;
    buff = buffer;
    for (k=0;k<(FRAME_WIDTH*FRAME_HEIGHT);k++) {
        if ((float)*buff < avg+threshold*sd)
            *buff = 0;
        else {
            *buff = *buff - avg;
            num++;
        }
        buff++;
    }
    sprintf (mess, "background avg=%f sd=%f num=%d\r\n", avg, sd, num);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();
    return (num);
}

void CCal3Dlg::OnOnePolRadio()
{
    pstates = 1;
}

void CCal3Dlg::OnAllPolRadio()
{
    pstates = POL_STATES;
```

```
}

void CCal3Dlg::OnOutfileBrowseButton()
{
    int iresult;

    ofn3.lStructSize = sizeof (OPENFILENAME);


    ofn3.hInstance = NULL;
    ofn3.hwndOwner = NULL;
    ofn3.lpstrFilter = "CTISP cal files (*.cal)\0*.cal\0All Files (*.*)\0*.*\0\0";
    ofn3.lpstrCustomFilter = NULL;
    ofn3.nMaxCustFilter = 0;
    ofn3.nFilterIndex = 1;
    ofn3.lpstrDefExt = "cal";
    ofn3.lCustData = NULL;
    ofn3.lpfnHook = NULL;
    ofn3.lpTemplateName = NULL;
    ofn3.lpstrFile = out_name;
    ofn3.nMaxFile = 500;
    ofn3.lpstrFileTitle = out_title;
    ofn3.nMaxFileTitle = 99;
    ofn3.lpstrInitialDir = "\\ctisp\\data";
    ofn3.lpstrTitle = "Open Output File";
    out_name[0] = '\0';
    iresult = GetOpenFileName (&ofn3);
    if (iresult) {
        UpdateData (TRUE);
        m_outfile = out_name;
        UpdateData (FALSE);
        UpdateWindow ();
    }
}
```

camera.cpp

```
#include "stdafx.h"
#include "camera.h"

int camera_setup ( BOOL bUseHighGain, BOOL bUseROI, int nMessageMode,
double
                exp_time)
{
  int  nResult;
  //char  szLastError[64];

  // Handle errors here
  pvSetErrorMode( PV_EM_SILENT );

  // Reset the board
  nResult = pvInitCapture( BOARDNUM );
  if ( nResult != SUCCESS )
  {
    printf("Error resetting board!\n" );
    goto fail;
  }

  // Set the device driver size expectations
  nResult = pvSetOptions( BOARDNUM, CCD_WIDTH, CCD_HEIGHT,
BITS_PER_PIXEL,
                        TIME_OUT, NUM_CHANNELS );
  if ( nResult != SUCCESS )
  {
    printf("Error setting device driver information!\n" );
    goto fail;
  }

  // Set the DLL size expectations
  nResult = pvSetCCDSize( BOARDNUM, CCD_WIDTH, CCD_HEIGHT );
  if ( nResult != SUCCESS )
  {
    printf("Error setting image size!\n" );
    goto fail;
  }

  // Set the PROM Page
  nResult = pvSetPROMPage( BOARDNUM,( bUseHighGain ? HIGH_GAIN :
LOW_GAIN ) );
  if ( nResult != SUCCESS )
  {
    printf("Error setting PROM page!\n" );
    goto fail;
  }
```

```
  // Set the CCD Temperature.  This should happen right after Set PROM Page,
  // because setting the PROM page reset the temperature to a default value
  nResult = pvSetCCDTemperatureCalibrated( BOARDNUM, CCD_TEMP );
  if ( nResult != SUCCESS )
  {
    printf("Error setting CCD temperature!\n" );
    goto fail;
  }

  // Set the camera timing constants.  The Master Clock, Serial Wait, and Parallel
Wait
  //   values are of particular interest because they determine the accuracy of
  //   the exposure time.
  // If you call pvSetWaitTimes, PVAPI will perform some calculations and call
this
  //   function anyway.  It's best to call this directly if you know the values.
  nResult = pvSetWaitConstants( BOARDNUM,
                        MASTER_CLOCK,
                        DISKING_WAIT,
                        PARALLEL_WAIT,
                        AFTER_EXPO,
                        SERIAL_WAIT,
                        SKIP_WAIT );
  if ( nResult != SUCCESS )
  {
    printf("Error setting timing constants!\n" );
    goto fail;
  }

  // Set the binning
  nResult = pvSetXBinning( BOARDNUM, X_BINNING );
  if ( nResult != SUCCESS )
  {
    printf("Error setting serial binning!\n" );
    goto fail;
  }

  nResult = pvSetYBinning( BOARDNUM, Y_BINNING );
  if ( nResult != SUCCESS )
  {
    printf("Error setting parallel binning!\n" );
    goto fail;
  }

  // Set the ROI or lack thereof
  if ( bUseROI )
  {
    nResult = pvEnableSingleROI( BOARDNUM, ROI_LEFT, ROI_TOP,
ROI_RIGHT,
```

```
                                    ROI_BOTTOM );
    if ( nResult != SUCCESS )
    {
      printf("Error setting region-of-interest!\n" );
      goto fail;
    }
  }
  else
  {
    nResult = pvDisableROI( BOARDNUM );
    if ( nResult != SUCCESS )
    {
      printf("Error disabling region-of-interest!\n" );
      goto fail;
    }
  }

  // Set the exposure time
  nResult = pvSetExposureMode( BOARDNUM, PV_XM_INTERNAL,
exp_time);
  if ( nResult != SUCCESS )
  {
    printf("Error setting exposure mode!\n" );
    goto fail;
  }

  return SUCCESS;

fail:
  if ( nMessageMode != NO_MESSAGES )
  {
    if ( nMessageMode == VERBOSE_MESSAGES )
    {
      // See if the last return code tells us more
      switch ( nResult )
      {
        case ERROR_NO_DRIVER:
          printf("The VxD could not be loaded.\n" );
          break;

        case ERROR_SERIAL_INPUT_LINK_BAD:
          printf("The input link is not connected.\n" );
          break;

        case ERROR_SERIAL_LINK_BAD:
          printf("\n\nThe output serial link is not connected." );
          break;

        case ERROR_SERIAL_NO_RESPONSE:
          printf("\n\nThe camera did not respond to the serial command." );
```

```
            break;

          case ERROR_SERIAL_BAD_RESPONSE:
            printf("\n\nAn unexpected serial response was received." );
            break;

          case ERROR_SERIAL_WRITE_ERROR:
            printf("\n\nAn error occurred while writing to the serial port." );
            break;

          case ERROR_SERIAL_READ_ERROR:
            printf( "\n\nAn error occurred while reading from the serial port." );
            break;

          case ERROR_SERIAL_CANT_OPEN_PORT:
            printf( "\n\nAn error occurred while opening the serial port." );
            break;

          case ERROR_SERIAL_PORT_INIT_ERROR:
            printf( "\n\nAn error occurred while initializing the serial port." );
            break;

          default:
            break;
        }
      }

    }

    return nResult;
}
```

monochrometer.cpp

```
//this program controls the CVI CM110 monochromator using the marshallsoft
library of
// COM port functions and a COM port.  Written by Brian Miles
// IN THE CURRENT CONFIG THE PROGRAM NEEDS THE WSC32.LIB
FILE AS PART OF THE PROJECT

#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include "Sioerror.h"
#include "wsc.h"

//prototypes
void CheckStatusByte(int);
void ReadCom(int);
void cdecl SioError(int, char);
void SendChar(int,int);
int QueryMono(int,int);
void ClearBuffers(int);
void SelectGrating(int,int);
void GoToWavelength(int);
int MonochromatorSetup();
void ShutdownMonochromator();
void cdecl SioError(int Code, char *Text);

// global declarations
int ComNum;
int ichar;
BOOL EchoToComp;

int monochrometer_setup()
{
    int CompBaudRate,MonoBaudRate;
    BOOL FatalError=FALSE;

//which com port do you want to use?
    ComNum=COM4;

//initialization of com port
    //val1=SioReset(ComNum,512,512);
    if (SioReset(ComNum,512,512)<0)
SioError(SioReset(ComNum,512,512),"");
    else printf("COM%i initialized correctly\n",(ComNum+1));  //COMS start at
'0', thus add 1 to display right #

//   Set Computer baud rate to 9600 to initially talk to mono (that's its default
rate)
```

```
    CompBaudRate=9600;
    if (SioBaud(ComNum,CompBaudRate)<0)
SioError(SioBaud(ComNum,CompBaudRate),"");
    else printf("Computer Baud rate now set to %i baud\n",CompBaudRate);

//set flow control to hardware
    if (SioFlow(ComNum,'H')<0) SioError(SioFlow(ComNum,'H'),"");

//first clear the transmit and receive buffers
    ClearBuffers(ComNum);

//must set RTS high before receiving data from mono, leave it high
    if (SioRTS(ComNum,'S')<0) SioError(SioRTS(ComNum,'S'),"");

// Send command to Mono to change to 300 baud
    MonoBaudRate=300;  //this is purely for printing, it is set via ochar=58,05
    SendChar(ComNum,58);
    SendChar(ComNum,5);
    printf("Mono Baud rate now set to %i baud\n",MonoBaudRate);
    ReadCom(ComNum); //read value returned from monochromator

//  Now Set Computer's ComNum's baud rate to 300
    CompBaudRate=300;
    if (SioBaud(ComNum,CompBaudRate)<0)
SioError(SioBaud(ComNum,CompBaudRate),"");
    else printf("Computer Baud rate now set to %i baud\n",CompBaudRate);

//Check ECHO to verify monochromator is listening and ready!!!
    ClearBuffers(ComNum);
    SendChar(ComNum,27);
    ReadCom(ComNum);
    if (EchoToComp!=TRUE)
        {
        MessageBox (NULL, "THE MONOCHROMATOR IS NOT TALKING
!!!", "Calibrate",
                    MB_OK);
        return (-1);
        }

//Finish setting up the  monochromator
    //define size which is the size of the default step
    ClearBuffers(ComNum);
    SendChar(ComNum,55);
    SendChar(ComNum,5);
    ReadCom(ComNum);
    //Select the 1200 grating as default  ********** change this to 600nm
on 10/6/98 for higher bandpass *
    ClearBuffers(ComNum);
    SelectGrating(ComNum,600);
    //Set the units to Nanometers
```

```
    ClearBuffers(ComNum);
    SendChar(ComNum,50);
    SendChar(ComNum,01);
    ReadCom(ComNum);
    //Set the diffraction order to positive
    ClearBuffers(ComNum);
    SendChar(ComNum,51);
    SendChar(ComNum,01);
    ReadCom(ComNum);

    return (0);
} //end of program

void CheckStatusByte(int ByteToCheck)
{
    //if that character is a status byte examine it!!!
    //see pg 42 of Buchanan, Applied PC graphics, interrupts for this bitmask
operation...
    if (ByteToCheck!=27 & ByteToCheck!=24)
    {
        if (ByteToCheck & 128)
        {
            puts("Command NOT accepted!!!");
            if (ByteToCheck & 16) puts("Scan is Negative going!");
            else puts("Scan is positive going!");
            if (ByteToCheck & 32) puts("Specifier was too small!");
            else puts("Specifier was too large!");
        }
    }
}

void ReadCom(int CN)
{
    int NumBytesRec;

    Sleep(50);
    //find out how many bytes are in the retreive queue
    NumBytesRec=SioRxQue(CN);
    if (NumBytesRec<0) SioError(NumBytesRec,"");
    //read the characters in
    for (int nc=1;nc<=NumBytesRec;nc++)
    {
        //retrieve a character from the monochromator
        ichar=SioGetc(CN);
        if (ichar<0) SioError(ichar,"");
        if (ichar==27) EchoToComp=TRUE;
        CheckStatusByte(ichar);
    }
}
```

```
int QueryMono(int CN, int QueryByte)
{
    int c1,c2,c3,NumBytesRec;   //chars to be read in.
    ClearBuffers(CN); //clear receive and transmitt buffers
    SendChar(CN,56);//send initial common querry byte

    switch (QueryByte)
       {
       case 00:
          SendChar(CN,00);
          NumBytesRec=SioRxQue(CN);
          while (NumBytesRec<=0)
             {
             //
             //must set RTS high before receiving data from mono, leave it high
             if (SioRTS(CN,'S')<0) SioError(SioRTS(CN,'S'),"");
             //Check ECHO to verify monochromator is listening and ready!!!
             ClearBuffers(CN);
             SendChar(CN,27);
             ReadCom(CN);
             if (EchoToComp!=TRUE)
                {
                puts("!!!!!!!!!! THE MONOCHROMATOR IS NOT TALKING
!!!!!!!!!!!!!");
                //FatalError=TRUE;
             //  goto ENDPROG;
                }
             else
                puts("The monochromator is talking");
             //
             ClearBuffers(CN);
             SendChar(CN,00);
             NumBytesRec=SioRxQue(CN);
             c1=SioGetc(CN);
             if (c1<=0) SioError(c1,"");
             c2=SioGetc(CN);
             if (c2<=0) SioError(c2,"");
             Sleep(1000);
             }
          c1=SioGetc(CN);
          if (c1<=0) SioError(c1,"");
          c2=SioGetc(CN);
          if (c2<=0) SioError(c2,"");
          Sleep(1000);
          printf("Grating Position = %i nm\n",(256*c1+c2));
          return(256*c1+c2);
          break;
       case 01:
          SendChar(CN,01);
          c1=SioGetc(CN);
```

```
    c2=SioGetc(CN);
    switch (c2)
        {
        case 0:
            puts("Single");
            return(0);
            break;
        case 1:
            puts("Additive dbl");
            return(1);
            break;
        case 254:
            puts("Subtractive dbl");
            return(254);
            break;
        }
case 02:
    SendChar(CN,02);
    c1=SioGetc(CN);
    c2=SioGetc(CN);
    printf("Grating Resolution is = %i grooves/mm\n",(256*c1+c2));
    return(256*c1+c2);
    break;
case 03:
    SendChar(CN,03);
    c1=SioGetc(CN);
    c2=SioGetc(CN);
    printf("Grating Blaze is = %i nm\n",(256*c1+c2));
    return(256*c1+c2);
    break;
case 04:
    SendChar(CN,04);
    c1=SioGetc(CN);
    c2=SioGetc(CN);
    printf("Current Grating is number %i\n",c2);
    return(c2);
    break;
case 05:
    SendChar(CN,05);
    c1=SioGetc(CN);
    c2=SioGetc(CN);
    printf("Current Scan Speed is = %i \n",(256*c1+c2));
    return(256*c1+c2);
    break;
case 06:
    SendChar(CN,06);
    c1=SioGetc(CN);
    c2=SioGetc(CN);
    printf("Size byte is %i\n",c2);
    return(c2);
```

```
            break;
        case 13:
            SendChar(CN,13);
            c1=SioGetc(CN);
            c2=SioGetc(CN);
            printf("%i gratings installed\n",c2);
            return(c2);
            break;
        case 14:
            SendChar(CN,14);
            c1=SioGetc(CN);
            c2=SioGetc(CN);
            switch (c2)
                {
                case 0:
                    puts("Units are centi-microns");
                    return(0);
                    break;
                case 1:
                    puts("Units are nanometers");
                    return(1);
                    break;
                case 2:
                    puts("Units are Angstroms");
                    return(2);
                    break;
                }
        default:
            puts("QUERY INVALID !!!!!!!!!!");
            return(-1);
        }
//check for final status bytes from mono
c3=SioGetc(CN);
CheckStatusByte(c3);
c3=SioGetc(CN);
if (c3!=24)
    {
    puts("Did NOT receive final 24 status byte from mono");
    printf("value of final status bytes was %i\n",c3);
    }
}

void SendChar(int CN, int CharToSend)
{
    int CodeNum;

    if (SioCTS(CN)>0)   //CN is COM num
        {
            Sleep(100);//this seems like a good value thus far
            CodeNum=SioPutc(CN,CharToSend);
```

```
            if (CodeNum<0) SioError(CodeNum,"");
            //else printf("Sent %i to Mono\n",CharToSend);
            Sleep(200);//likewise for this sleep param.
        }
}

void ClearBuffers(int CN)
{
    Sleep(25);
    if (SioTxClear(CN)<0) SioError(SioTxClear(CN),"");
    if (SioRxClear(CN)<0) SioError(SioRxClear(CN),"");
    Sleep(25);
}

void SelectGrating(int CN, int Grating)
{
int val;

val=QueryMono(CN,04);//find out which number grating is installed 600=#2,
1200=#1
if (Grating==600)
    {
    ClearBuffers(CN);
    SendChar(CN,26);
    SendChar(CN,2);
    ReadCom(CN);
    puts("600 g/mm grating selected");
    if (val!=2) Sleep(5000); //wait for grating to change if 600 was not selected
previously
    else Sleep(2000);
    }
else
    {
    ClearBuffers(CN);
    SendChar(CN,26);
    SendChar(CN,1);
    puts("1200 g/mm grating selected");
    if (val!=1) Sleep(5000); //wait for grating to change if 600 was not selected
previously
    else Sleep(2000);
    }
if (Grating!=600 & Grating!=1200) puts("WARNING INVALID GRATING
SELECTED");
}

void GoToWavelength(int Wavelength)
{
int HighByte,LowByte, OldWavelength;
double time2wait;
int CN;
```

```
CN = ComNum;
//determine wait based on how far we have to move
Sleep(75);
ClearBuffers(CN);
OldWavelength=QueryMono(CN,00);
printf("Old Wavelength is %i\n",OldWavelength);
ClearBuffers(CN);

//determine high and low byte
HighByte=Wavelength/256;
LowByte=Wavelength-HighByte*256;
SendChar(CN,16);
SendChar(CN,HighByte);
SendChar(CN,LowByte);
ReadCom(CN);
time2wait=fabs(OldWavelength-(double)Wavelength)/100;
printf("Waiting %.1f seconds for grating to move to %i
nm\n",time2wait,Wavelength);
Sleep((long)time2wait*1000);
 //allow time for grating to move, 1 sec per 100nm
}
```

pol_states.cpp

```
#include "stdafx.h"
#include "cal3.h"
#include "cal3Dlg.h"

double v[100];
int pd[100];
int cur_pos;

void CCal3Dlg::polar_cal ()
{
    int i, iresp;
    int min, max;
    double minv, maxv;
    double a, x, y;
    int npts;
    char mess[200];
    int iResult;

    printf ("Performing polarizer calibration\n");
    // Flip mirror up and set waveplate to 5 volts
    Flip_Mirror (up);
  set_wp_voltage (5.0);

    // Rotate turnstage one complete revolution and store detector responses
    max = 0;
    min = 99999999;
    npts = STEPS_PER_REV / STEP_INC + 1;
    for (i=0; i<=npts; i++) {
        v[i] = i*STEP_INC;
        rotate_polarizer ((int)v[i], ABS_MODE);
        pd[i] = read_photodetector ();
        if (pd[i] > max) {
            max = pd[i];
            maxv = v[i];
        }
        if (pd[i] < min) {
            min = pd[i];
            minv = v[i];
        }
    }
    sprintf (mess, "minpos %f min %d maxpos %f max %d\r\n", minv, min,
maxv, max);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();

    // Find the position that gives vertical polarization
```

```
    best_fit (max-min, max-min, 1, 0, 202000, 203, (double)min, (double)min, 1,
&a, &x, &y,
            COSINE, npts);
    best_fit (a-(a*0.1), a+(a*0.1), 21, x-1000, x+1000, 21, y-(y*0.1), y+(y*0.1),
21, &a, &x,
            &y, COSINE, npts);
    best_fit (a-(a*0.01), a+(a*0.01), 21, x-100, x+100, 21, y-(y*0.01),
y+(y*0.01), 21, &a, &x,
            &y, COSINE, npts);
    best_fit (a-(a*0.001), a+(a*0.001), 21, x-10, x+10, 21, y-(y*0.001),
y+(y*0.001), 21, &a,
            &x, &y, COSINE, npts);

    sprintf (mess, "Best fit parameters: a = %f  x = %f  y = %f\r\n", a, x, y);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();

    // Rotate polarizer to max position
    rotate_polarizer ((int)x, ABS_MODE);
    iresp = read_photodetector ();
    printf ("max pos = %d  Det resp = %d\n", (int)x, iresp);

    // Reset stepper motor so that current position is 0
    polarizer_reset ();

    // Flip mirror back down
    Flip_Mirror (down);
}


double CCal3Dlg::circular_pol(int wavelength)
{
    int i;
    double minv, maxv;
    double a, x, y;
    int min, max;
    double pvoltage;
    int npts;
    char mess[200];
    int iResult;

    // Calculate predicted voltage at minimum

    pvoltage = -0.00159568*wavelength + 3.82863592;
    sprintf (mess, "waveplate voltage at %dnm is %f\r\n", wavelength, pvoltage);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();
    return (pvoltage);
```

```
}


void best_fit (double alow, double ahigh, int asteps, double xlow, double xhigh,
int xsteps,
            double ylow, double yhigh, int ysteps, double *a, double *x, double
*y,
            int curve_type, int npts)
{
    double minerr, sumerr;
    double xinc, yinc, ainc;
    int i, j, k, m;
    double x0, y0, a0;
    double err;

    minerr = 99999999999999.0;
    if (xsteps > 1)
        xinc = (xhigh-xlow) / (xsteps-1);
    else
        xinc = 0;
    if (ysteps > 1)
        yinc = (yhigh-ylow) / (ysteps-1);
    else
        yinc = 0;
    if (asteps > 1)
        ainc = (ahigh-alow) / (asteps-1);
    else
        ainc = 0;
//  printf ("xlow=%f ylow=%f xinc=%f, yinc=%f\n", xlow, ylow, xinc, yinc);

    for (i=0; i<xsteps; i++) {
        x0 = xlow + i*xinc;
        for (j=0; j<ysteps; j++) {
            y0 = ylow + j*yinc;
            for (k=0; k<asteps; k++) {
                a0 = alow + k*ainc;
                sumerr = 0.0;
                for (m=0; m<npts; m++) {
                    if (curve_type == PARABOLA)
                        err = pd[m] - (y0+a0*(v[m]-x0)*(v[m]-x0));
                    else
                        err = pd[m] - (y0+a0*cos((v[m]-
x0)/STEPS_PER_REV*2.0*3.1459)*
                            cos((v[m]-x0)/STEPS_PER_REV*2.0*3.1459));
                    sumerr += err*err;
                }
                    if (sumerr < minerr) {
                    minerr = sumerr;
                    *a = a0;
```

```
                    *y = y0;
                    *x = x0;
                }
            }
        }
    }
    printf ("Min error = %f\n", minerr);
}


int waveplate_setup ()
{
    i16 RetVal;
    i16 IgnoreWarning;
    i16 status;

    // Configure port 0
    status = DIG_Prt_Config (DEVICE1, port0, MODE, dir);
    RetVal=NIDAQErrorHandler(status,"DIG_Prt_Config ",IgnoreWarning);
    if (status != 0) {
        return (status);
    }

    // Configure port 1
    status = DIG_Prt_Config (DEVICE1, port1, MODE, dir);
    RetVal=NIDAQErrorHandler(status,"DIG_Prt_Config ",IgnoreWarning);
    if (status != 0) {
        return (status);
    }

    return (0);
}


int photodetector_setup ()
{
    i16 RetVal;
    i16 IgnoreWarning;
    i16 status;

    //configure input line for photodetector
    status=AI_Configure(DEVICE2,Channel0,InputMode,InputRange,polarity,
                driveAIS);
    RetVal=NIDAQErrorHandler(status,"AI_Configure",IgnoreWarning);
    return (status);
}


void rotate_polarizer (int value, int mode)
```

```
{
    int iresult;
    char s[100];
    int wait;

    if (mode == ABS_MODE) {
        iresult = SioPuts (POL_PORT, "IAB*\r", 5);
        wait = (int)(fabs((float)value-(float)cur_pos) / FEEDRATE * 1000 +
250);
        cur_pos = value;
    }
    else if (mode == IN_MODE) {
        iresult = SioPuts (POL_PORT, "IIN*\r", 5);
        wait = (int)(fabs((float)value) / (float)FEEDRATE * 1000 + 250);
        cur_pos += value;
    }
    else {
        printf ("Mode is incorrect\n");
        exit (-1);
    }
    Sleep (250);

    sprintf (s, "IXF%5dD%7d*\r", FEEDRATE, value);
    iresult = SioPuts (POL_PORT, s, 18);
//  printf ("Current polarizer position: %d\n", cur_pos);
    Sleep (wait);
}


void CCal3Dlg::set_wp_voltage (float volts)
{
    int value;
    i16 pattern0, pattern1;
    i16 status;
    i16 RetVal;
    i16 IgnoreWarning;

//  printf ("Setting wave plate voltage to %f\n", volts);

    // Convert voltage to two 8-bit values
    value = (int)(volts / voltage_res);
    if (value > 32767)
        value = 32767;
    pattern1 = (value / 256) | 128;
    pattern0 = value % 256;

    // Write low-order byte to port 0
    status = DIG_Out_Port (DEVICE1, port0, pattern0);
    RetVal=NIDAQErrorHandler(status,"DIG_Out_Port",IgnoreWarning);
    if (status != 0) {
```

```
        printf ("Error: status %d writing to port 0\n", status);
        exit (-1);
    }

    // Write high-order byte to port 1
    status = DIG_Out_Port (DEVICE1, port1, pattern1);
    RetVal=NIDAQErrorHandler(status,"DIG_Out_Port",IgnoreWarning);
    if (status != 0) {
        printf ("Error: status %d writing to port 1\n", status);
        exit (-1);
    }
    Sleep (500);
    m_cur_wp = volts;
    UpdateData(FALSE);
    update_status_scroll ();
}


int read_photodetector ()
{
    double Voltage, VoltageSum;
    int mean;
    i16 status;
    i16 RetVal;
    i16 IgnoreWarning;

    //read photodetector signal multiple times
    VoltageSum=0;
    for (int j=0; j<NREADS; j++) {
        status=AI_VRead(DEVICE2,Channel0,Gain,&Voltage);
        RetVal=NIDAQErrorHandler(status,"AI_VRead",IgnoreWarning);
        VoltageSum=VoltageSum+Voltage;
    }

    // Average responses and convert to microvolts
    mean = (int)(VoltageSum * 1000000 / NREADS + 0.5);

    return (mean);
}



void polarizer_setup ()
{
    int CompBaudRate;
    BOOL FatalError=FALSE;

    //initialization of com port
    if (SioReset(POL_PORT,1024,512)<0)
```

```
SioError(SioReset(POL_PORT,1024,512),"");
    else printf("COM%i initialized correctly\n",(POL_PORT+1));   //COMS start
at '0', thus add 1 to display right #

    //  Set Computer baud rate to 9600 (that's its default rate)
    CompBaudRate=9600;
    if (SioBaud(POL_PORT,CompBaudRate)<0)
SioError(SioBaud(POL_PORT,CompBaudRate),"");
    else printf("Computer Baud rate now set to %i baud\n",CompBaudRate);

    //first clear the transmit and receive buffers
    ClearBuffers(POL_PORT);

    //must set RTS high before receiving data from stage, leave it high
    if (SioDTR(POL_PORT,'S')<0) SioError(SioDTR(POL_PORT,'S'),"");
    printf ("DTR set high\n");
    if (SioRTS(POL_PORT,'S')<0) SioError(SioRTS(POL_PORT,'S'),"");
    printf("RTS set high\n");

    //set flow control to software (xon, xoff)
    if (SioFlow(POL_PORT,'N')<0) SioError(SioFlow(POL_PORT,'N'),"");

    Sleep(1000);
}

void pol_filter_setup()
{
    int CompBaudRate;
    BOOL FatalError=FALSE;
    int iresult;

    //initialization of com port
    if (SioReset(POL_FILTER_PORT,1024,512)<0)
        SioError(SioReset(POL_FILTER_PORT,1024,512),"");
    else printf("COM%i initialized correctly\n",(POL_FILTER_PORT+1));
//COMS start at '0', thus add 1 to display right #

    //  Set Computer baud rate to 9600
    CompBaudRate=9600;
    if (SioBaud(POL_FILTER_PORT,CompBaudRate)<0)
        SioError(SioBaud(POL_FILTER_PORT,CompBaudRate),"");
    else printf("Computer Baud rate now set to %i baud\n",CompBaudRate);

    // Set communications parameters the same as stepper driver for now
(9600,8,1,none) ...
    if (SioParms(POL_FILTER_PORT,NoParity,OneStopBit,WordLength8)<0)

    SioError(SioParms(POL_FILTER_PORT,NoParity,OneStopBit,WordLength
8),"");
    else printf("COM%i parameters changed\n",(POL_FILTER_PORT+1));
```

```
    //first clear the transmit and receive buffers
    ClearBuffers(POL_FILTER_PORT);

    //must set RTS high before receiving data from stage, leave it high
    SioDTR(POL_FILTER_PORT,'S');
    if ((iresult = SioRTS(POL_FILTER_PORT,'S'))<0)
        SioError(SioRTS(POL_FILTER_PORT,'S'),"");//was 'R'
    printf ("RTS = %d\n", iresult);

    //set flow control to software (xon, xoff)
    if (SioFlow(POL_FILTER_PORT,'S')<0)
        SioError(SioFlow(POL_FILTER_PORT,'S'),"");

    Sleep(1000);

    //SEND THE ATTENTION COMMAND

    SioPuts(POL_FILTER_PORT," ",1);
    Sleep (1000);
    SioPuts (POL_FILTER_PORT, "\r",1);
    Sleep (1000);

    //enter setup commands for stepper
    iresult = SioPuts (POL_FILTER_PORT, "Y 5 25\r",7);
    Sleep (1000);
    iresult = SioPuts (POL_FILTER_PORT, "H 0\r",4);  //set steps to constant
step size
    Sleep (1000);
    iresult = SioPuts (POL_FILTER_PORT, "D 8\r",4); //set step resolution to
D8 1/256 step
    Sleep (1000);
    iresult = SioPuts (POL_FILTER_PORT, "e 400\r",6);//set encoder lines/rev
to 400
    Sleep (1000);
    iresult = SioPuts (POL_FILTER_PORT, "I 100\r",6); //set initial velocity to
100
    Sleep (1000);
    iresult = SioPuts (POL_FILTER_PORT, "V 350\r",6); //set slew velocity to
450
    Sleep (1000);
    iresult = SioPuts (POL_FILTER_PORT, "K 50 50\r",8); //set ramp
acceleration 50/50
    Sleep (1000);
//  find_home_filter ();
}

void find_home_filter ()
{
    int iresult;
```

```
    char s[200];
    int move_mode, nchars;
    char mess[200];

    iresult = SioPuts (POL_FILTER_PORT, "F 350 1\r",8); // find home
    Sleep (500);
    nchars = SioGets (POL_FILTER_PORT, s, 200);
    sprintf (mess, "nchars=%d", nchars);
        iresult = MessageBox (NULL, mess, "Calibrate", MB_OKCANCEL);
        if (iresult == IDCANCEL)
            exit (-1);
    Sleep (500);
    move_mode = 8;
    while ((move_mode & 8) == 8) {
       iresult=SioPuts (POL_FILTER_PORT, "^\r", 2);
       Sleep (500);
       nchars = SioGets (POL_FILTER_PORT, s, 100);
       s[nchars-1] = '\0';
       sscanf (&s[2], "%d", &move_mode);
          sprintf (mess, "move status=%d", move_mode);
          iresult = MessageBox (NULL, mess, "Calibrate", MB_OKCANCEL);
          if (iresult == IDCANCEL)
              exit (-1);
    }
    iresult = SioPuts (POL_FILTER_PORT, "O\r", 2); // reset origin
    Sleep (1000);
    rotate_pol_filter (-360);
    iresult = SioPuts (POL_FILTER_PORT, "O\r", 2); // reset origin
    Sleep (1000);
}

void rotate_pol_filter (int pos)
{
    int iresult;
    char s[100];
    int move_mode, nchars;
    int newpos;
    char mess[200];

    nchars = SioGets (POL_FILTER_PORT, s, 100);
    Sleep (500);
    newpos = -1;
    while (newpos != pos) {
       sprintf (s, "R+%4d\r", pos);
       iresult = SioPuts (POL_FILTER_PORT, s, 7);
       Sleep (1000);
       nchars = SioGets (POL_FILTER_PORT, s, 100);

       move_mode = 1;
       while ((move_mode % 2) > 0) {
```

```
            iresult=SioPuts (POL_FILTER_PORT, "^\r", 2);
            Sleep (500);
            nchars = SioGets (POL_FILTER_PORT, s, 100);
            s[nchars-1] = '\0';
            sscanf (&s[2], "%d", &move_mode);
        }
        SioPuts (POL_FILTER_PORT, "z\r", 2);
        Sleep (500);
        nchars = SioGets (POL_FILTER_PORT, s, 100);
        s[nchars-1] = '\0';
        sscanf (&s[2], "%d", &newpos);
    }
}


void polarizer_reset ()
{
    int iresult;

    iresult = SioPuts (POL_PORT, "\4", 1);
    Sleep (1000);
    iresult = SioPuts (POL_PORT, ">>01\r", 5);
    cur_pos = 0;
    Sleep (1000);
}


void com_close (int CN)

{
    //Close the COM port
    if (SioDone(CN)<0)
        SioError(SioDone(CN),"");
    else
        printf("COM%i closed successfully\n",(CN+1));
}


int Flip_Mirror(i16 UpOrDown)
{
    const i16 DeviceNumber=2;//currently the pci board is configured by nidaq config to device 2
    const i16 PCIPort=0;        //all E series boards use port 0
    const i16 FlipperLine=0;  //Digital io line for mirror flipper
    const i16 Input_or_Output=1; //set line 0 to be output
    const i16 GoHigh=1;         //sets logic state to high
    const i16 GoLow=0;          //sets logic state to low
    const i16 up=1;
    const i16 down=0;

    i16 Status=0;
```

```
    i16 RetVal=0;
    i16 IgnoreWarning=0;

    //configure  digital line for output ....
    Status=DIG_Line_Config(DeviceNumber,PCIPort,FlipperLine,Input_or_Out
put);
    RetVal=NIDAQErrorHandler(Status,"DIG_Line_Config",IgnoreWarning);

    switch (UpOrDown)
    {
    case up:
        {
        Status=DIG_Out_Line(DeviceNumber,PCIPort,FlipperLine,GoHigh);
        RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
        printf ("Mirror up\n");
        }
        break;
    case down:
        {
        Status=DIG_Out_Line(DeviceNumber,PCIPort,FlipperLine,GoHigh);
        RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
        Sleep(50);
        printf ("Mirror down\n");
        Status=DIG_Out_Line(DeviceNumber,PCIPort,FlipperLine,GoLow);
        RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
        }
        break;
    default:
        {
        printf("invalid argument for flipper, must be up or down\n");
        return(-1);
        }
    }

    Sleep (1500);
    return(0);
}

int find_vertical_pol ()
{
    const i16 DeviceNumber=2;//currently the pci board is configured by nidaq config to device 2
    const i16 PCIPort=0;       //all E series boards use port 0
    const i16 Line=7;   //Digital io line for mirror flipper
    const i16 Input_or_Output=0; //set line 0 to be input
     i16 state;
    const step_size = 200;
    int i;

    i16 Status=0;
    i16 RetVal=0;
```

```
    i16 IgnoreWarning=0;

    //configure  digital line for input ....
    Status=DIG_Line_Config(DeviceNumber,PCIPort,Line,Input_or_Output);
    RetVal=NIDAQErrorHandler(Status,"DIG_Line_Config",IgnoreWarning);

    Status=DIG_In_Line(DeviceNumber,PCIPort,Line, &state);
    RetVal=NIDAQErrorHandler(Status,"DIG_In_Line",IgnoreWarning);
    if (state == 1) return (0);
    rotate_polarizer(-10*step_size, ABS_MODE);
    polarizer_reset ();
    for (i=0; i<STEPS_PER_REV; i+=step_size) {
        Status=DIG_In_Line(DeviceNumber,PCIPort,Line, &state);
        RetVal=NIDAQErrorHandler(Status,"DIG_In_Line",IgnoreWarning);
        if (state == 1) break;
        rotate_polarizer(i, ABS_MODE);
    }

    Sleep (1500);
    return(0);
}
```

xyposition.cpp

```
#include "stdafx.h"
#include "cal3.h"
#include "cal3Dlg.h"

struct target tgt[MAXTGTS];
int hotpix[14]={54435, 130676,112503,363408,338223,286037,719206,301221,
                    541468,452421,771616,748032,709048,713117};

// move_fiber - moves the fiber to dersired location
//
void CCal3Dlg::move_fiber (unsigned short int *buffer, int xguess, int yguess,
                           int xptime, int xmtime, int yptime, int ymtime,
                           float tolerance, int update_move_rates)
{
    float xdiff, ydiff;
    fiber_pos *prev_pos;
    int duration;
    int iResult;
    float pixels_moved;
    struct stats_st image_stats;
    int nmoves;
    char mess[200];

    prev_pos = pos;
    xdiff = xguess - pos->x;
    ydiff = yguess - pos->y;
    sprintf (mess, "xguess=%d yguess=%d posx=%f posy=%f xdiff=%f
ydiff=%f\r\n",
        xguess, yguess, pos->x, pos->y, xdiff, ydiff);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();
    while ((fabs(xdiff) > tolerance) || (fabs(ydiff) > tolerance)) {

        // Move in x direction
        nmoves = 0;
        while ((fabs(xdiff) > tolerance) && (nmoves < max_moves)) {
            if (xdiff > 0.0) {
                duration = (int)(xdiff * xptime);
                if (duration > max_xduration)
                    duration = max_xduration;
                sprintf (mess, "xdiff = %f duration = %d \r\n", xdiff, duration);
                m_Status_Edit += mess;
                UpdateData (FALSE);
                update_status_scroll ();
                iResult = Move_X (Xplus, duration);
            }
```

```
            else if (xdiff < 0.0) {
                duration = (int)((-xdiff) * xmtime);
                if (duration > max_xduration)
                    duration = max_xduration;
                sprintf (mess, "xdiff = %f duration = %d \r\n", xdiff, duration);
                m_Status_Edit += mess;
                UpdateData (FALSE);
                update_status_scroll ();
                iResult = Move_X (Xminus, duration);
            }
            iResult=pvAcquireFrame(BOARDNUM,buffer);
            if (iResult!=0) printf("ERROR pvacquireframe returned
%i\n",iResult);
            stats (buffer, &image_stats);
            pos = find_center   (buffer, image_stats.mean*thresh);
            pixels_moved = fabs(prev_pos->x - pos->x);
            sprintf (mess, "pixmoved = %f\r\n", pixels_moved);
            m_Status_Edit += mess;
            UpdateData (FALSE);
            update_status_scroll ();

            xdiff = (float)xguess - pos->x;
            free(prev_pos);
            prev_pos = pos;
            nmoves++;
        }

        // Move in y direction

        ydiff = yguess - pos->y;
        nmoves = 0;
        while ((fabs(ydiff) > tolerance) && (nmoves < max_moves)) {
            if (fabs(ydiff) < tolerance*2.0)
                ydiff = ydiff/1.5;
            if (ydiff > 0.0) {
                duration = (int)(ydiff * yptime);
                if (duration > max_yduration)
                    duration = max_yduration;
                sprintf (mess, "ydiff = %f duration = %d \r\n", ydiff, duration);
                m_Status_Edit += mess;
                UpdateData (FALSE);
                update_status_scroll ();
                iResult = Move_Y (Yplus, duration);
            }
            else if (ydiff < 0.0) {
                duration = (int)((-ydiff) * ymtime);
                if (duration > max_yduration)
                    duration = max_yduration;
                sprintf (mess, "ydiff = %f duration = %d \r\n", ydiff, duration);
                m_Status_Edit += mess;
```

```
                UpdateData (FALSE);
                update_status_scroll ();
                iResult = Move_Y (Yminus, duration);
            }
            iResult=pvAcquireFrame(BOARDNUM,buffer);
            if (iResult!=0) printf("ERROR pvacquireframe returned
%i\n",iResult);
            stats (buffer, &image_stats);
            pos = find_center   (buffer, image_stats.mean*thresh);
            pixels_moved = fabs(prev_pos->y - pos->y);
            if (update_move_rates == 1 && pixels_moved > MIN_YMOVE) {
                yptime = duration / pixels_moved;
                ymtime = yptime;
            }
            sprintf (mess, "pixmoved=%f\r\n", pixels_moved);
            m_Status_Edit += mess;
            UpdateData (FALSE);
            update_status_scroll ();
            ydiff = (float)yguess - pos->y;

            free(prev_pos);
            prev_pos = pos;
            nmoves++;
        }
        xdiff = (float)xguess - pos->x;
        ydiff = (float)yguess - pos->y;
        xdiff = 0.0;
//      printf ("xdiff = %f ydiff = %f\n", xdiff, ydiff);
    }
    return;
}


void CCal3Dlg::xmove_cal (unsigned short int *buffer, int *xptime, int
*xmtime)
{
    int iResult;
    float xdiff, ydiff;
    fiber_pos *prev_pos;
    struct stats_st image_stats;
    char mess[200];

    // Move in positive x direction
    //
    free (pos);
    iResult = Move_X (Xplus, init_xduration/10);
    iResult=pvAcquireFrame(BOARDNUM,buffer);
    if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
    stats (buffer, &image_stats);
    prev_pos = find_center    (buffer, image_stats.mean*thresh);
```

```
iResult = Move_X (Xplus, init_xduration);
iResult=pvAcquireFrame(BOARDNUM,buffer);
if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
stats (buffer, &image_stats);
pos = find_center   (buffer, image_stats.mean*thresh);
xdiff = pos->x - prev_pos->x;
ydiff = pos->y - prev_pos->y;
if (xdiff < 0.0) {
    MessageBox ("Fiber position moved in wrong direction for plus X",
                "Calibrate", MB_OK);
    OnExitButton ();
}
*xptime = (int)(init_xduration/xdiff);
sprintf(mess, "xdiff = %5.2f ydiff = %5.2f plusxtime = %d\r\n", xdiff,
        ydiff, *xptime);
m_Status_Edit += mess;
m_plusx = *xptime;
UpdateData (FALSE);
update_status_scroll ();
if (m_plusx < MIN_XMOVE_TIME || m_plusx > MAX_XMOVE_TIME) {
    sprintf (mess, "Plus X move time = %d", m_plusx);
    iResult = MessageBox (mess, "Calibrate", MB_OKCANCEL);
    if (iResult == IDCANCEL)
        OnExitButton ();
}

// Move in negative x direction
//
free(prev_pos);
free(pos);
iResult = Move_X (Xminus, init_xduration/10);
iResult=pvAcquireFrame(BOARDNUM,buffer);
if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
stats (buffer, &image_stats);
prev_pos = find_center    (buffer, image_stats.mean*thresh);
iResult = Move_X (Xminus, init_xduration);
iResult=pvAcquireFrame(BOARDNUM,buffer);
if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
stats (buffer, &image_stats);
pos = find_center   (buffer, image_stats.mean*thresh);
xdiff = pos->x - prev_pos->x;
ydiff = pos->y - prev_pos->y;
if (xdiff > 0.0) {
    MessageBox ("Fiber position moved in wrong direction for minus X",
                "Calibrate", MB_OK);
    OnExitButton ();
}
*xmtime = abs((int)(init_xduration/xdiff));
sprintf(mess, "xdiff = %5.2f ydiff = %5.2f minusxtime = %d\r\n", xdiff,
        ydiff, *xmtime);
```

```
    m_Status_Edit += mess;
    m_minusx = *xmtime;
    UpdateData (FALSE);
    update_status_scroll ();
    if (m_minusx < MIN_XMOVE_TIME || m_minusx >
MAX_XMOVE_TIME) {
        sprintf (mess, "Minus X move time = %d", m_minusx);
        iResult = MessageBox (mess, "Calibrate", MB_OKCANCEL);
        if (iResult == IDCANCEL)
            OnExitButton ();
    }
    return;
}

void CCal3Dlg::ymove_cal (unsigned short int *buffer, int *yptime, int
*ymtime)
{
    int iResult;
    float xdiff, ydiff;
    fiber_pos *prev_pos;
    struct stats_st image_stats;
    char mess[200];
    // Move in positive y direction
    //
    free(pos);
    iResult = Move_Y (Yplus, init_yduration/10);
    iResult=pvAcquireFrame(BOARDNUM,buffer);
    if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
    stats (buffer, &image_stats);
    prev_pos = find_center    (buffer, image_stats.mean*thresh);
    iResult = Move_Y (Yplus, init_yduration);
    iResult=pvAcquireFrame(BOARDNUM,buffer);
    if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
    stats (buffer, &image_stats);
    pos = find_center   (buffer, image_stats.mean*thresh);
    xdiff = pos->x - prev_pos->x;
    ydiff = pos->y - prev_pos->y;
    if (ydiff < 0.0) {
        MessageBox ("Fiber position moved in wrong direction for plus Y",
                "Calibrate", MB_OK);
        OnExitButton ();
    }
    *yptime = (int)(init_yduration/ydiff);
    sprintf(mess, "xdiff = %5.2f ydiff = %5.2f plusytime = %d\r\n", xdiff,
            ydiff, *yptime);
    m_Status_Edit += mess;
    m_plusy = *yptime;
    UpdateData (FALSE);
    update_status_scroll ();
    if (m_plusy < MIN_YMOVE_TIME || m_plusy > MAX_YMOVE_TIME) {
```

```
        sprintf (mess, "Plus Y move time = %d", m_plusy);
        iResult = MessageBox (mess, "Calibrate", MB_OKCANCEL);
        if (iResult == IDCANCEL)
            OnExitButton ();
    }

    // Move in negative y direction
    //
    free(prev_pos);
    free(pos);
    iResult = Move_Y (Yminus, init_yduration/10);
    iResult=pvAcquireFrame(BOARDNUM,buffer);
    if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
    stats (buffer, &image_stats);
    prev_pos = find_center    (buffer, image_stats.mean*thresh);
    iResult = Move_Y (Yminus, init_yduration);
    iResult=pvAcquireFrame(BOARDNUM,buffer);
    if (iResult!=0) printf("ERROR pvacquireframe returned %i\n",iResult);
    stats (buffer, &image_stats);
    pos = find_center   (buffer, image_stats.mean*thresh);
    xdiff = pos->x - prev_pos->x;
    ydiff = pos->y - prev_pos->y;
    if (ydiff > 0.0) {
        MessageBox ("Fiber position moved in wrong direction for minus Y",
                "Calibrate", MB_OK);
        OnExitButton ();
    }
    *ymtime = abs((int)(init_yduration/ydiff));
    sprintf(mess, "xdiff = %5.2f ydiff = %5.2f minusytime = %d\r\n", xdiff,
            ydiff, *ymtime);
    m_Status_Edit += mess;
    m_minusy = *ymtime;
    UpdateData (FALSE);
    update_status_scroll ();
    free(prev_pos);
    if (m_minusy < MIN_YMOVE_TIME || m_minusy >
MAX_YMOVE_TIME) {
        sprintf (mess, "Minus Y move time = %d", m_minusy);
        iResult = MessageBox (mess, "Calibrate", MB_OKCANCEL);
        if (iResult == IDCANCEL)
            OnExitButton ();
    }

    return;
}

//
//
// This procedure finds the current location of the fiber.
//
```

```
fiber_pos *CCal3Dlg::find_center (unsigned short *buffer, float threshold)
{
    int i, j, k;
    int index, maxpts;
    int width;
    struct point *pnt;
    int iresult, check;
    int ntgts, val;
    fiber_pos *pos;
    float xw, yw;
    char mess[200];

    // Allocate storage for position structure
    //
    pos = (fiber_pos *)malloc(sizeof(fiber_pos));

    // Find all objects consisting of pixels above threshold
    //
    width = FRAME_WIDTH;
    ntgts = 0;
    for (i=MINYFOV-ROI_TOP; i <= MAXYFOV-ROI_TOP; i++) {
        for (j=MINXFOV-ROI_LEFT; j<=MAXXFOV-ROI_LEFT; j++) {
            val = *(buffer+j+(i*width));
            if (val > threshold) {

                // Fill in point structure
                pnt = (struct point *)malloc(sizeof(struct point));
                pnt->x = j;
                pnt->y = i;
                pnt->val = val;

                // Determine whether this point is adjacent to an existing object.
                // If so, merge it with the object.
                iresult = 0;
            for (k=ntgts-1; k >= 0; k--)
                    if (iresult = compare(pnt, &tgt[k])) {
                merge (pnt, &tgt[k]);
                break;
                    }

                // If not merged with an existing object, create a new one
            if (!iresult) {
               init_tgt (pnt, &tgt[ntgts]);
               ntgts++;
                  }
             }
        }
    }

    // Compare all objects with each other and merge any that are contiguous
```

```
    for (i=0; i<ntgts; i++) {
        if (tgt[i].mean >= 0) {
            check = 1;
            while (check) {
                check = 0;
                for (j=i+1; j<ntgts; j++)
                    if (iresult = comp_tgts(&tgt[i], &tgt[j])) {
                        merge_tgts (&tgt[i], &tgt[j]);
                        tgt[j].mean = -1;
                        fprintf (stderr, "merged targets %d and %d\n", i, j);
                        check = 1;
                    }
            }
        }
    }

    // Find object with the largest number of points. This should be the
    // fiber image.
    if (ntgts == 0) {
        MessageBox ("No fiber image detected", "Calibrate", MB_OK);
        OnExitButton ();
    }
    maxpts = 0;
    for (i=0; i<ntgts; i++)
        if (tgt[i].npts > maxpts) {
            maxpts = tgt[i].npts;
            index = i;
        }

    // Fill position structure
    pos->x = tgt[index].xc;
    pos->y = tgt[index].yc;
    pos->intensity = (int)(tgt[index].mean + 0.5);
    pos->npts = tgt[index].npts;

    // Add ROI offset to center coordinates
    //
    pos->x += ROI_LEFT;
    pos->y += ROI_TOP;
    m_xpos = pos->x;
    m_ypos = pos->y;
    UpdateData (FALSE);
    update_status_scroll ();

    return (pos);
}


void init_tgt (struct point *pnt, struct target *targ)
```

```
{
  targ->xc = (float)pnt->x;
  targ->yc = (float)pnt->y;
  targ->xcw = (float)pnt->x * (float)pnt->x;
  targ->ycw = (float)pnt->y * (float)pnt->y;
  targ->minx = pnt->x;
  targ->maxx = pnt->x;
  targ->miny = pnt->y;
  targ->maxy = pnt->y;
  targ->npts = 1;
  targ->mean = (float)pnt->val;
  targ->p = pnt;
  pnt->nextp = NULL;
}


int compare(struct point *pnt, struct target *targ)

{
  struct point *nxp;

  if ((pnt->x >= (targ->minx-MAXD)) && (pnt->x <= (targ->maxx)+MAXD)
&&
     (pnt->y >= (targ->miny-MAXD)) && (pnt->y <= (targ->maxy)+MAXD)) {
    nxp = targ->p;
    while (nxp != NULL) {
      if ((abs(nxp->x - pnt->x) <= MAXD) && (abs(nxp->y - pnt->y) <=
MAXD))
        return (1);
      nxp = nxp->nextp;
    }
  }
  return (0);
}


void merge (struct point *pnt, struct target *targ)

{
  pnt->nextp = targ->p;
  targ->p = pnt;
  targ->xc = (targ->xc*targ->npts+pnt->x) / (targ->npts+1);
  targ->yc = (targ->yc*targ->npts+pnt->y) / (targ->npts+1);
  targ->xcw += pnt->x * pnt->val;
  targ->ycw += pnt->y * pnt->val;
  targ->mean = (targ->mean*targ->npts+pnt->val) /
               (targ->npts+1);
  targ->npts++;
  if (pnt->x < targ->minx) targ->minx = pnt->x;
  if (pnt->x > targ->maxx) targ->maxx = pnt->x;
```

```
  if (pnt->y < targ->miny) targ->miny = pnt->y;
  if (pnt->y > targ->maxy) targ->maxy = pnt->y;
}


int comp_tgts(struct target *targ1, struct target *targ2)

{
  struct point *nxp1, *nxp2;

  if (targ2->mean == -1)
      return (0);
  if ((targ1->maxx < (targ2->minx-MAXD)) || (targ1->minx > (targ2-
>maxx)+MAXD) ||
    (targ1->maxy < (targ2->miny-MAXD)) || (targ1->miny > (targ2-
>maxy)+MAXD))
    return (0);

  nxp1 = targ1->p;
  while (nxp1 != NULL) {
    nxp2 = targ2->p;
    while (nxp2 != NULL) {
      if ((abs(nxp2->x - nxp1->x) <= MAXD) && (abs(nxp2->y - nxp1->y) <=
MAXD))
        return (1);
      nxp2 = nxp2->nextp;
    }
    nxp1 = nxp1->nextp;
  }
  return (0);
}


void merge_tgts (struct target *targ1, struct target *targ2)
{
  struct point *nxp;

  nxp = targ1->p;
  while (nxp->nextp != NULL) nxp = nxp->nextp;
  nxp->nextp = targ2->p;
  targ1->xc = (targ1->xc*targ1->npts+targ2->xc*targ2->npts) /
        (targ1->npts+targ2->npts);
  targ1->yc = (targ1->yc*targ1->npts+targ2->yc*targ2->npts) /
        (targ1->npts+targ2->npts);
  targ1->xcw += targ2->xcw;
  targ1->ycw += targ2->ycw;
  targ1->mean = (targ1->mean*targ1->npts+targ2->mean*targ2->npts) /
            (targ1->npts+targ2->npts);
  targ1->npts += targ2->npts;
  if (targ2->minx < targ1->minx) targ1->minx = targ2->minx;
```

```
  if (targ2->maxx > targ1->maxx) targ1->maxx = targ2->maxx;
  if (targ2->miny < targ1->miny) targ1->miny = targ2->miny;
  if (targ2->maxy > targ1->maxy) targ1->maxy = targ2->maxy;
}

//
// stats - calculates stats for an image
//
void CCal3Dlg::stats (unsigned short *buffer, struct stats_st *image_stats)
{
    int k;
    float BufferSum=0;
    float avg;
    unsigned short *buff;
    int imin, imax;

    // Set hot pixels in CCD array to zero
    //
    for (k=0; k<14; k++)
        *(buffer+hotpix[k]) = 0;

    imax=*buffer;
    imin=*buffer;
    buff = buffer;
    BufferSum = 0;
    for (k=0;k<(FRAME_WIDTH*FRAME_HEIGHT);k++) {
        BufferSum += *buff;
        if (*buff>imax) imax=*buff;
        if ((*buff<imin) && (*buff > 0)) imin=*buff;
        buff++;
    }
    avg = BufferSum/(FRAME_WIDTH*FRAME_HEIGHT);
    image_stats->min = imin;
    image_stats->max = imax;
    image_stats->mean = avg;
    m_imin = imin;
    m_imax = imax;
    m_imean = avg;
    UpdateData (FALSE);
    update_status_scroll ();
}


int Move_Y(int Direction, int nsteps)
{
    i16 Status=0;
    i16 RetVal=0;
    i16 IgnoreWarning=0;
    int i;
```

```
    //configure digital lines 1,2 for output as direction and step ....
    Status=DIG_Line_Config(YDEVICE,PCIPort,YDirectionLine,Input_or_Out
put);
    RetVal=NIDAQErrorHandler(Status,"DIG_Line_Config",IgnoreWarning);
    Status=DIG_Line_Config(YDEVICE,PCIPort,YStepLine,Input_or_Output);
    RetVal=NIDAQErrorHandler(Status,"DIG_Line_Config",IgnoreWarning);

    switch( Direction )
    {
  case Yplus:
     {
        Status=DIG_Out_Line(YDEVICE,PCIPort,YDirectionLine,Yplus);
        RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
        printf(" ... Moving in PLUS Y direction for %i steps\n",nsteps);
        for (i=0; i<nsteps; i++) {
           Status=DIG_Out_Line(YDEVICE,PCIPort,YStepLine,GoHigh);

    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
           Sleep (10);
           Status=DIG_Out_Line(YDEVICE,PCIPort,YStepLine,GoLow);

    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
           if (i<num_accel_steps) Sleep(int(num_accel_steps/(i+1)));
           if ((i>num_accel_steps) && (i>(nsteps-num_accel_steps)))
Sleep(int(num_accel_steps/(nsteps-i+1)));
        }
        }
     break;
  case Yminus:
        {
        Status=DIG_Out_Line(YDEVICE,PCIPort,YDirectionLine,Yminus);
        RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
        printf(" ... Moving MINUS Y direction for %i steps\n",nsteps);.....
        for (i=0; i<nsteps; i++) {
           Status=DIG_Out_Line(YDEVICE,PCIPort,YStepLine,GoHigh);

    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
           Sleep (10);
           Status=DIG_Out_Line(YDEVICE,PCIPort,YStepLine,GoLow);

    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
           if (i<num_accel_steps) Sleep(int(num_accel_steps/(i+1)));
           if ((i>num_accel_steps) && (i>(nsteps-num_accel_steps)))
Sleep(int(num_accel_steps/(nsteps-i+1)));
        }
        }
     break;
  default :
     {
        printf("ERROR!!! invalid direction chosen\n");
```

```
        return(-1);
        }
    }
    return(0);
}

int Move_X(int Direction, int nsteps)
{
    int i;
    i16 Status=0;
    i16 RetVal=0;
    i16 IgnoreWarning=0;

    //configure digital lines 3,4 for output as direction and step ....
    Status=DIG_Line_Config(XDEVICE,PCIPort,XDirectionLine,Input_or_Out
put);
    RetVal=NIDAQErrorHandler(Status,"DIG_Line_Config",IgnoreWarning);
    Status=DIG_Line_Config(XDEVICE,PCIPort,XStepLine,Input_or_Output);
    RetVal=NIDAQErrorHandler(Status,"DIG_Line_Config",IgnoreWarning);

    switch( Direction )
    {
  case Xplus:
     {
        Status=DIG_Out_Line(XDEVICE,PCIPort,XDirectionLine,Xplus);
        RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
        printf(" ... Moving in PLUS X direction %i steps\n",nsteps);
        for (i=0; i<nsteps; i++) {
            Status=DIG_Out_Line(XDEVICE,PCIPort,XStepLine,GoLow);

    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
            Status=DIG_Out_Line(XDEVICE,PCIPort,XStepLine,GoHigh);

    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
            if (i<num_accel_steps) Sleep(int(num_accel_steps/(i+1)));
            if ((i>num_accel_steps) && (i>(nsteps-num_accel_steps)))
Sleep(int(num_accel_steps/(nsteps-i+1)));
        }
        }
     break;
  case Xminus:
        {
        Status=DIG_Out_Line(XDEVICE,PCIPort,XDirectionLine,Xminus);
        RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
        printf(" ... Moving in MINUS X direction %i steps\n",nsteps);
        for (i=0; i<nsteps; i++) {
            Status=DIG_Out_Line(XDEVICE,PCIPort,XStepLine,GoLow);

    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
            Status=DIG_Out_Line(XDEVICE,PCIPort,XStepLine,GoHigh);
```

```
    RetVal=NIDAQErrorHandler(Status,"DIG_Out_Line",IgnoreWarning);
            if (i<num_accel_steps) Sleep(int(num_accel_steps/(i+1)));
            if ((i>num_accel_steps) && (i>(nsteps-num_accel_steps)))
Sleep(int(num_accel_steps/(nsteps-i+1)));
          }
          }
      break;
   default :
      {
        printf("ERROR!!! invalid direction chosen\n");
        return(-1);
        }
     }
     return(0);
}
```

sioerror.c

```
#include "wsc.h"
#include "paint.h"

void cdecl SioError(int Code, char *Text)
{static char Temp[80];
#ifdef WIN32
 DWORD dwError;
 char  *Win32ErrMsg;
#endif
 puts(Text);
 switch(Code)
   {case WSC_NO_DATA:
     puts("No Data");
     break;
    case WSC_RANGE:
     puts("Parameter out of range");
     break;
    case WSC_ABORTED:
     puts("Aborted");
     break;
    case WSC_EXPIRED:
     puts("Shareware execution expired");
     break;
    case IE_BADID:
     puts(" Invalid COM port\n");
     break;
    case IE_OPEN:
     puts(" COM port already open\n");
     break;
    case IE_NOPEN:
     puts(" Cannot open COM port\n");
     break;
    case IE_MEMORY:
     puts(" Cannot allocate memory\n");
     break;
    case IE_DEFAULT:
     puts(" Error in default parameters\n");
     break;
    case IE_HARDWARE:
     puts(" COM port hardware not present\n");
     break;
    case IE_BYTESIZE:
     puts(" Unsupported byte size\n");
     break;
    case IE_BAUDRATE:
     puts(" Unsupported baud rate\n");
     break;
```

```
#ifdef WIN32
   case WSC_WIN32ERR:
    dwError = (DWORD) SioWinError();

if(FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM|FORMAT_MESS
AGE_ALLOCATE_BUFFER|80 ,
                NULL, dwError,
                MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US),
                (LPTSTR) &Win32ErrMsg, 0, NULL)
          > 0)
       {puts(Win32ErrMsg);
       }
     else
       {sprintf(Temp," Win32 Error\n");
        puts(Temp);
       }
     break;
#endif
   default:
     sprintf(Temp," Unknown code %d\n",Code);
     puts(Temp);
     break;
  }
} /* end SioError */
```

cal3.h

```
// cal3.h : main header file for the CAL3 application
//

#if
!defined(AFX_CAL3_H__84BAE265_E02C_11D1_8216_0000C0A97971__IN
CLUDED_)
#define
AFX_CAL3_H__84BAE265_E02C_11D1_8216_0000C0A97971__INCLUDED
_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////
// CCal3App:
// See cal3.cpp for the implementation of this class
//

class CCal3App : public CWinApp
{
public:
    CCal3App();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCal3App)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CCal3App)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_CAL3_H__84BAE265_E02C_11D1_8216_0000C0A97971__IN
CLUDED_)
```

cal3Dlg.h

```
// cal3Dlg.h : header file
//

#if
!defined(AFX_CAL3DLG_H__84BAE267_E02C_11D1_8216_0000C0A97971
__INCLUDED_)
#define
AFX_CAL3DLG_H__84BAE267_E02C_11D1_8216_0000C0A97971__INCLU
DED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <math.h>
#include "camera.h"
#include "monochromator.h"
#include "param.h"
#include "pol_states.h"
#include "Sioerror.h"
#include "wsc.h"
#include "xyposition.h"
/////////////////////////////////////////////////////////////////////////////
// CCal3Dlg dialog

class CCal3Dlg : public CDialog
{
// Construction
public:
    CCal3Dlg(CWnd* pParent = NULL);.....// standard constructor

// Dialog Data
    //{{AFX_DATA(CCal3Dlg)
    enum { IDD = IDD_CAL3_DIALOG };
    CComboBox    m_pol_con;
    CEdit m_Status_Con;
    UINT m_winc;
    UINT m_wstart;
    UINT m_wsteps;
    UINT m_xstart;
    UINT m_ystart;
    CString   m_pstate;
    UINT m_wave;
    UINT m_imax;
```

```
    UINT m_imean;
    UINT m_imin;
    UINT m_minusx;
    UINT m_minusy;
    UINT m_plusx;
    UINT m_plusy;
    CString   m_Status_Edit;
    float   m_xpos;
    float   m_ypos;
    CString   m_pol;
    float   m_iexp;
    CString   m_outfile;
    UINT m_cur_exp;
    UINT m_tot_exp;
    float   m_cur_wp;
    float   m_exp;
    //}}AFX_DATA

    // User defined functions and data
    int pstates;
    int zero (unsigned short *, float, int, int);
    fiber_pos *pos;
    OPENFILENAME ofn3;
    char out_name[500], out_title[100];

    void update_status_scroll ();
    float set_exposure (unsigned short *, float);
    void polar_cal ();
    double circular_pol (int);
    void set_wp_voltage (float);
    void xmove_cal (unsigned short int *, int *, int *);
    void ymove_cal (unsigned short int *, int *, int *);
    void move_fiber (unsigned short int *, int, int, int, int, int, int, float, int);
    void stats (unsigned short *, struct stats_st *);
    fiber_pos *find_center (unsigned short *, float);

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCal3Dlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CCal3Dlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
```

```
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnCalButton();
    afx_msg void OnExitButton();
    afx_msg void OnOnePolRadio();
    afx_msg void OnAllPolRadio();
    afx_msg void OnOutfileBrowseButton();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_CAL3DLG_H__84BAE267_E02C_11D1_8216_0000C0A97971
__INCLUDED_)
```

camera.h

```
extern "C"
{
#include "pvapi.h"
}
#include "pvapi.h"

const int NO_MESSAGES     = 0;
const int SHORT_MESSAGES  = 1;
const int VERBOSE_MESSAGES = 2;
const unsigned short CCD_WIDTH     = 1040;
const unsigned short CCD_HEIGHT    = 1024;
const unsigned short BITS_PER_PIXEL = 16;
const unsigned char  BOARDNUM      = 0;
const unsigned int    TIME_OUT      = 10000;//milliseconds
const unsigned short HIGH_GAIN     = 0;   //PROM page of high gain setting
const unsigned short LOW_GAIN      = 4;   //PROM page of low gain setting
const unsigned short NUM_CHANNELS  = 1;   //number of channels (1,2 or 4)
const unsigned short CCD_TEMP      = 285;  //CCD temp in Kelvin
const unsigned short X_BINNING     = 1;   //Bin this many pixels in X direction
const unsigned short Y_BINNING     = 1;   //Bin this many pixels in Y direction
const unsigned short MASTER_CLOCK  = 625;  //Clock frequency is 62.5 ns
const unsigned short DISKING_WAIT  = 11;   //The camera converts this to a
time
const unsigned short PARALLEL_WAIT = 795;  //"  "     "       "    " " "
const unsigned short AFTER_EXPO    = 375;  //"  "     "       "    " " "
const unsigned short SERIAL_WAIT   = 2;    //"  "     "       "    " " "
const unsigned short SKIP_WAIT     = 2;    //"  "     "       "    " " "
const unsigned short ROI_LEFT      = 15; .... //use15,1038,0,1023 for full display
const unsigned short ROI_RIGHT     = 1038;
const unsigned short ROI_TOP       = 0;
const unsigned short ROI_BOTTOM    = 1023;
const unsigned short FRAME_WIDTH   = (ROI_RIGHT-ROI_LEFT+1);
const unsigned short FRAME_HEIGHT  = (ROI_BOTTOM-ROI_TOP+1);
const int MAX_INTENSITY          = 65535;
//const double EXPOSURE_TIME       = 0.03;    //exposure time in seconds
const int MINXFOV                = 435;
const int MAXXFOV                = 555;
const int MINYFOV                = 456;
const int MAXYFOV                = 574;
int camera_setup (BOOL, BOOL, int, double);
```

param.h

```
const int xstart                = 515;
const int ystart                = 530;
const int xsteps                = 1;
const int ysteps                = 1;
const int xinc                  = 5;
const int yinc                  = 5;
const int wstart                = 450;
const int winc                  = 50;
const int wsteps                = 5;
const int POL_STATES                    = 4;
const float thresh              = 1.5;
const int init_wave             = 650;
const float intensity_thresh            = 0.9;
```

pol_states.h

```
#include "nidaq.h"
#include "nidaqerr.h"
#include "nidaqex.h"
#include "wsc.h"

const i16 DEVICE1 = 1;
const i16 port0=0;
const i16 port1=1;
const i16 port2=2;
const i16 MODE=0;
const i16 dir=1;
const float voltage_res=0.00061;
const i16 DEVICE2=2;   //device number for XE50 board
const i16 Channel0=0;
const i16 InputMode=1; //referenced single ended
const i16 InputRange=0;
const i16 polarity=0;  //bipolar operation
const i16 driveAIS=1;  //drive AIsense to ground
const i16 Gain=1; //1,2,10,100 valid choices
const int NREADS=1000;
const int AVG_PTS=7;
const int FEEDRATE=10000;
const int IN_MODE=0;
const int ABS_MODE=1;
const int LOCKIN_PORT=4;
const int POL_PORT=COM3;
const int POL_FILTER_PORT=COM2;
const int PARABOLA=0;
const int COSINE=1;
const int STEPS_PER_REV=202500;
const int STEP_INC=2500;
const i16 up=1;
const i16 down=0;
const int POL_FILTER_OFFSET=0;
const int STEPS_PER_FILTER=320;

double circular_pol (int);
void best_fit (double, double, int, double, double, int, double, double, int,
        double *, double *, double *, int, int);
void polarizer_reset ();
void com_close (int);
void ClearBuffers (int);
void polarizer_setup ();
void pol_filter_setup ();
void rotate_pol_filter (int);
void find_home_filter ();
int waveplate_setup ();
```

```
int photodetector_setup ();
void rotate_polarizer (int, int);
void set_wp_voltage (float);
int read_photodetector ();
int Flip_Mirror(i16);
int find_vertical_pol ();
```

xyposition.h

```
#include "nidaqex.h"
#include "nidaq.h"
#include "nidaqerr.h"   //may not need this

#define  MAXD      1
#define  MAXTGTS   100
const i16 XDEVICE=2;    //nidaq pci board
const i16 YDEVICE=2;    //nidaq pci board
const i16 PCIPort=0;        //all E series boards use port 0
const i16 XDirectionLine=3;  //stepper motor direction input for X movement
const i16 YDirectionLine=1;  //stepper motor direction input..... for Y movement
const i16 XStepLine=4;        //stepper motor step input for X movement
const i16 YStepLine=2;        //stepper motor step input for Y movement
const i16 Input_or_Output=1;//set lines 0-3 to be output
const i16 GoHigh=1;           //sets logic state to high
const i16 GoLow=0;            //sets logic state to low
const i16 Xplus=1;          //increasing X standpoint of image
const i16 Xminus=0;         //decreasing X "
const i16 Yplus=1;          //increasing Y standpoint of image
const i16 Yminus=0;         //decreasing Y "
const float loc_tolerance=0.5;
const int max_moves=5;
const int num_accel_steps=50;  //acceleration parameter for stepper motors
const int init_xduration=5000; // in steps
const int init_yduration=500; //     "
const int max_xduration=10000; //     "
const int max_yduration=1000; //      "
const int xwidth=80;
const int ywidth=80;
const int MIN_XMOVE_TIME=300;
const int MAX_XMOVE_TIME=1500;
const int MIN_YMOVE_TIME=30;
const int MAX_YMOVE_TIME=150;
const float MIN_YMOVE=2.0;
const int MAXY_POS=20;
/*
struct move_time {
        float total_pix;
        int total_time;
        int time;
};
*/
typedef struct {
        int npts;
        int intensity;
        float x;
        float y;
```

```
} fiber_pos;

struct point {
     int  x;
     int  y;
     int  val;
     struct point *nextp;
};

struct target {
     float xc;
     float yc;
       float xcw;
       float ycw;
     int  minx;
     int  maxx;
     int  miny;
     int  maxy;
     int  npts;
     float mean;
     struct point *p;
};

struct stats_st {
          int  min;
          int  max;
          float mean;
};

int Move_Y(int, int);
int Move_X(int, int);
void init_tgt (struct point *, struct target *);
int compare(struct point *, struct target *);
void merge (struct point *, struct target *);
int comp_tgts(struct target *, struct target *);
void merge_tgts (struct target *, struct target *);
```

Acquire_Image

Files: acquire_image.cpp
acquire_imageDlg.cpp
acq.cpp
camera.cpp
sercomm.cpp
acquire_image.h
acquire_imageDlg.h
acq.h
camera.h
sercomm.h

acquire_image.cpp

```
// acquire_image.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "acquire_image.h"
#include "acquire_imageDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAcquire_imageApp

BEGIN_MESSAGE_MAP(CAcquire_imageApp, CWinApp)
    //{{AFX_MSG_MAP(CAcquire_imageApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAcquire_imageApp construction

CAcquire_imageApp::CAcquire_imageApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```

```
//////////////////////////////////////////////////////////////////////////
// The one and only CAcquire_imageApp object

CAcquire_imageApp theApp;

//////////////////////////////////////////////////////////////////////////
// CAcquire_imageApp initialization

BOOL CAcquire_imageApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();            // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();// Call this when linking to MFC statically
#endif

    CAcquire_imageDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

acquire_imageDlg.cpp

```
// acquire_imageDlg.cpp : implementation file
//

#include "stdafx.h"
#include "acquire_image.h"
#include "acquire_imageDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAcquire_imageDlg dialog

CAcquire_imageDlg::CAcquire_imageDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CAcquire_imageDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAcquire_imageDlg)
    m_fname = _T("");
    m_max = 0;
    m_min = 0;
    m_mean = 0;
    m_exp = 0.2f;
    m_temp = 290;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CAcquire_imageDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAcquire_imageDlg)
    DDX_Text(pDX, IDC_FILE_EDIT, m_fname);
    DDX_Text(pDX, IDC_MAX_EDIT, m_max);
    DDX_Text(pDX, IDC_MIN_EDIT, m_min);
    DDX_Text(pDX, IDC_MEAN_EDIT3, m_mean);
    DDX_Text(pDX, IDC_EXP_BOX, m_exp);
    DDX_Text(pDX, IDC_TEMP_BOX, m_temp);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAcquire_imageDlg, CDialog)
    //{{AFX_MSG_MAP(CAcquire_imageDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_ACQUIRE_BUTTON, OnAcquireButton)
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
```

```
    ON_BN_CLICKED(IDC_FILEBROWSE_BUTTON, OnFilebrowseButton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAcquire_imageDlg message handlers

BOOL CAcquire_imageDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);    // Set big icon
    SetIcon(m_hIcon, FALSE);  // Set small icon

    // TODO: Add extra initialization here

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CAcquire_imageDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
```

```
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CAcquire_imageDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CAcquire_imageDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CAcquire_imageDlg::OnAcquireButton()
{
    int iresult;
    BOOL bUseHighGain=1;
    BOOL bUseROI=1;
    int nMessageMode=NO_MESSAGES;
    unsigned short header[13];
    char mess[200];
    int i;
```

```
    // Get parameters
    //
    UpdateData (TRUE);
    strcpy (fname, (LPCTSTR)m_fname);

    // Make sure that the Frame width defined by the ROI is even.
    // An odd value will result in an image with line to line
    // horizontal shifts due to a bug in the camera software.
    if ((FRAME_WIDTH % 2) != 0) {
        MessageBox ("Frame width must be even. Adjust ROI parameters.",
                "Acquire_image", MB_OK);
        OnExitButton ();
    }

    // Set up camera
    iresult = camera_setup(bUseHighGain, bUseROI, nMessageMode, m_exp,
m_temp);
    if (iresult!=0) {
        sprintf(mess, "Error: camera_setup returned %i\n",iresult);
        MessageBox (mess, "Acquire_image", MB_OK);
        OnExitButton ();
    }

    // Set up polarization filters
    //
    pol_filter_setup();

    // Open output file
    //
    if( (outfile = fopen(fname,"wb" )) == NULL ) {
        MessageBox ("Error Opening Output File", "acquire_image", MB_OK);
        return;
    }

    // Write header
    //
    header[0] = NFILTERS;
    header[1] = FRAME_WIDTH;
    header[2] = FRAME_HEIGHT;
    for (i=0; i<NFILTERS; i++) {
        header[i+3] = i;
        header[i+3+NFILTERS] = m_exp*10000;
    }
    fwrite (header, 2, 13, outfile);

    //allocate buffer memory
    //
    buffer= (unsigned short *)calloc((FRAME_HEIGHT*FRAME_WIDTH),
 sizeof(unsigned short));
    if (buffer==NULL) {
```

```
        MessageBox("Can't allocate memory", "acquire_image", MB_OK);
        return;
    }

    // Acquire images
    //
    for (i=0; i<NFILTERS; i++) {
        rotate_pol_filter (i*STEPS_PER_FILTER+POL_FILTER_OFFSET);
        Sleep(2000);
        iresult = acquire (buffer);
    }
    rotate_pol_filter (POL_FILTER_OFFSET);
    MessageBox ("Done", "Acquire_image", MB_OK);
}

void CAcquire_imageDlg::OnExitButton()
{
    DestroyWindow ();
    exit (0);
}

void CAcquire_imageDlg::OnOK()
{
    return;
}

void CAcquire_imageDlg::OnFilebrowseButton()
{
    int iresult;

    ofn2.lStructSize = sizeof (OPENFILENAME);
    ofn2.hInstance = NULL;
    ofn2.hwndOwner = NULL;
    ofn2.lpstrFilter = "CTISP files (*.ctp)\0*.ctp\0All Files (*.*)\0*.*\0\0";
    ofn2.lpstrCustomFilter = NULL;
    ofn2.nMaxCustFilter = 0;
    ofn2.nFilterIndex = 1;
    ofn2.lpstrDefExt = "ctp";
    ofn2.lCustData = NULL;
    ofn2.lpfnHook = NULL;
    ofn2.lpTemplateName = NULL;
    ofn2.lpstrFile = fname;
    ofn2.nMaxFile = 500;
    ofn2.lpstrFileTitle = ftitle;
    ofn2.nMaxFileTitle = 99;
    ofn2.lpstrInitialDir = "\\ctisp\\data";
    ofn2.lpstrTitle = "Open Output File";
    fname[0] = '\0';
    iresult = GetOpenFileName (&ofn2);
    if (iresult) {
```

```
        UpdateData (TRUE);
        m_fname = fname;
        UpdateData (FALSE);
        UpdateWindow ();
    }
}
```

acq.cpp

```
//calibrate.cpp

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <math.h>
#include "stdafx.h"
#include "acq.h"
#include "acquire_image.h"
#include "acquire_imageDlg.h"

//prototypes
//
void stats (unsigned short *, struct stats_st *);

int CAcquire_imageDlg::acquire (unsigned short *buffer)
{
    int iResult;
    struct stats_st image_stats;


    // Acquire image
    //
    iResult=pvAcquireFrame(BOARDNUM,buffer);
    if (iResult!=0) {
        MessageBox ("ERROR acquiring frame", "acquire_image", MB_OK);
       return (-1);
    }

    // Calculate and display stats
    //
    stats (buffer, &image_stats);
    m_min = image_stats.min;
    m_max = image_stats.max;
    m_mean = image_stats.mean + 0.5;
    UpdateData (FALSE);
    UpdateWindow ();

    // Write image to file
    //
    fwrite (buffer, 2, FRAME_HEIGHT*FRAME_WIDTH, outfile); .....

    return(0);
}

//
// stats - calculates stats for an image
```

```
//
void stats (unsigned short *buffer, struct stats_st *image_stats)
{
    int k;
    float BufferSum=0;
    float avg;
    unsigned short *buff;
    int imin, imax;

    imax=*buffer;
    imin=*buffer;
    buff = buffer;
    BufferSum = 0;
    for (k=0;k<(FRAME_WIDTH*FRAME_HEIGHT);k++) {
        BufferSum += *buff;
        if (*buff>imax) imax=*buff;
        if ((*buff<imin) && (*buff > 0)) imin=*buff;
        buff++;
    }
    avg = BufferSum/(FRAME_WIDTH*FRAME_HEIGHT);
    image_stats->min = imin;
    image_stats->max = imax;
    image_stats->mean = avg;
}

void CAcquire_imageDlg::pol_filter_setup()
{
    DWORD CompBaudRate;
    BOOL FatalError=FALSE;
    int iresult;
//scp  serial comm port  extra type defs
   char CommPortName[6];
   COMMTIMEOUTS CommTimeOuts;
   DCB dcb;
   BOOL fRetVal ;

//scp  create I/O event used for overlapped reads/writes
  OL_Read_Pol.hEvent = CreateEvent(NULL,    // no security
                        TRUE,    // explicit reset req
                        FALSE,   // initial event reset
                        NULL); // no name
  if (OL_Read_Pol.hEvent == NULL)
  {
    printf("CreateEvent for Read Failed.\n");
    return;
  }
  OL_Write_Pol.hEvent = CreateEvent(NULL,    // no security
                        TRUE,    // explicit reset req
                        FALSE,   // initial event reset
                        NULL); // no name
```

```
  if (OL_Write_Pol.hEvent == NULL)
  {
    CloseHandle(OL_Read_Pol.hEvent);
    printf("CreateEvent for Write Failed.\n");
    return;
  }
//scp  initialize offsets for overlapped reads/writes
  OL_Read_Pol.Offset = 0 ;
  OL_Read_Pol.OffsetHigh = 0 ;
  OL_Write_Pol.Offset = 0 ;
  OL_Write_Pol.OffsetHigh = 0 ;

  //initialization of com port
//sio    if (SioReset(POL_FILTER_PORT,1024,512)<0)
//sio        SioError(SioReset(POL_FILTER_PORT,1024,512),"");
//scp  generate the COMM port name  (ie. COM1, COM2, ..., COM99)
//scp  COM1 is 1, COM2 is 2, ..., COM12 is 12, and are
//scp    defined in SerComm.h.  Add defines there for COM## upto 99
  wsprintf(CommPortName, "COM%d", POL_FILTER_PORT) ;
//scp  open COMM device
   if ((hPol_Filter_Port =
    CreateFile( CommPortName, GENERIC_READ | GENERIC_WRITE,
            0,              // exclusive access
            NULL,           // no security attrs
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL |
            FILE_FLAG_OVERLAPPED, // overlapped I/O
            NULL )) == (HANDLE) -1 )
    {
    printf("CreateFile for Comm. Port Failed.\n");
    return;
    }
   else
   {
//scp  get any early notifications
    SetCommMask(hPol_Filter_Port, EV_RXCHAR) ;
//scp  setup device buffers
//scp  went with 4096 on buffers from tty example
//scp    instead of 1024 and 512 used in sio call
    SetupComm(hPol_Filter_Port, 4096, 4096) ;
//scp  purge any information in the buffer
    PurgeComm(hPol_Filter_Port, PURGE_TXABORT | PURGE_RXABORT |
                    PURGE_TXCLEAR | PURGE_RXCLEAR ) ;
//scp  set up for overlapped I/O
    CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF ;
    CommTimeOuts.ReadTotalTimeoutMultiplier = 0 ;
    CommTimeOuts.ReadTotalTimeoutConstant = 1000 ;
//scp  CBR_9600 is approximately 1byte/ms. For our purposes, allow
    //   Set Computer baud rate to 9600
      CompBaudRate=9600;
```

```
//scp  double the expected time per character for a fudge factor.
    CommTimeOuts.WriteTotalTimeoutMultiplier =
2*CBR_9600/CompBaudRate;
    CommTimeOuts.WriteTotalTimeoutConstant = 0 ;
    SetCommTimeouts(hPol_Filter_Port, &CommTimeOuts ) ;
//sio     printf("COM%i initialized correctly\n",(POL_FILTER_PORT+1));
//COMS start at '0', thus add 1 to display right #
    printf("COM%i initialized correctly\n",(POL_FILTER_PORT)); //COM1 is
1, COM2 is 2, ...
  }
//scp  setup serial comm port via data control block (dcb)
  dcb.DCBlength = sizeof( DCB ) ;
//scp  get initial state from comm port
  GetCommState(hPol_Filter_Port, &dcb ) ;

//sio    if (SioBaud(POL_FILTER_PORT,CompBaudRate)<0)
//sio        SioError(SioBaud(POL_FILTER_PORT,CompBaudRate),"");
//sio    else printf("Computer Baud rate now set to %i baud\n",CompBaudRate);
  dcb.BaudRate = CompBaudRate;

    // Set communications parameters the same as stepper driver for now
(9600,8,1,none) ...
//sio    if
(SioParms(POL_FILTER_PORT,NoParity,OneStopBit,WordLength8)<0)
//sio
    SioError(SioParms(POL_FILTER_PORT,NoParity,OneStopBit,WordLength
8),"");
//sio    else printf("COM%i parameters changed\n",(POL_FILTER_PORT+1));
  dcb.ByteSize = 8;
  dcb.Parity = NOPARITY;
  dcb.StopBits = ONESTOPBIT;
//scp  setup flow control to software (xon, xoff)
//scp  disable DTRDSR handshaking
  dcb.fOutxDsrFlow = 0;
//scp  enables DTR line
  dcb.fDtrControl = DTR_CONTROL_ENABLE;
//scp  disable RTSCTS handshaking
    dcb.fOutxCtsFlow = 0;
//scp  enables RTS line
  dcb.fRtsControl = RTS_CONTROL_ENABLE;
//scp  enable XONXOFF software handshaking
  dcb.fInX = dcb.fOutX = 1;
  dcb.XonChar = ASCII_XON;
  dcb.XoffChar = ASCII_XOFF;
  dcb.XonLim = 100;
  dcb.XoffLim = 100;
//scp  other various settings
  dcb.fBinary = TRUE ;
  dcb.fParity = TRUE ;
//scp  set serial comm port via dcb
```

```
   fRetVal = SetCommState(hPol_Filter_Port, &dcb);
   printf("Computer Baud rate now set to %i baud\n",CompBaudRate);
   printf("COM%i parameters changed\n",POL_FILTER_PORT);

     //first clear the transmit and receive buffers
//sio    ClearBuffers(POL_FILTER_PORT);
//scp  already done right after buffer definition

     //must set RTS high before receiving data from stage, leave it high
//sio    SioDTR(POL_FILTER_PORT,'S');
   EscapeCommFunction(hPol_Filter_Port, SETDTR);
//sio    if ((iresult = SioRTS(POL_FILTER_PORT,'S'))<0)
//sio        SioError(SioRTS(POL_FILTER_PORT,'S'),"");//was 'R'
   iresult = int(EscapeCommFunction(hPol_Filter_Port, SETRTS));
     printf ("RTS = %d\n", iresult);

     //set flow control to software (xon, xoff)
//sio    if (SioFlow(POL_FILTER_PORT,'S')<0)
//sio        SioError(SioFlow(POL_FILTER_PORT,'S'),"");
//scp  already done in dcb

     Sleep(1000);

     //SEND THE ATTENTION COMMAND

//sio    SioPuts(POL_FILTER_PORT," ",1);
   WriteCommBlock(hPol_Filter_Port," ",1,&OL_Write_Pol);
     Sleep (1000);
//sio    SioPuts (POL_FILTER_PORT, "\r",1);
   WriteCommBlock(hPol_Filter_Port,"\r",1,&OL_Write_Pol);
     Sleep (1000);

     //enter setup commands for stepper
//sio    iresult = SioPuts (POL_FILTER_PORT, "Y 5 25\r",7);
     iresult = int(WriteCommBlock (hPol_Filter_Port,"Y 5
25\r",7,&OL_Write_Pol));
     Sleep (1000);
//sio    iresult = SioPuts (POL_FILTER_PORT, "H 0\r",4);  //set steps to
constant step size
     iresult = int(WriteCommBlock (hPol_Filter_Port,"H
0\r",4,&OL_Write_Pol));
     Sleep (1000);
//sio    iresult = SioPuts (POL_FILTER_PORT, "D 7\r",4); //set step resolution
to D6 1/128 step
     iresult = int(WriteCommBlock (hPol_Filter_Port,"D
7\r",4,&OL_Write_Pol));
     Sleep (1000);
//sio    iresult = SioPuts (POL_FILTER_PORT, "e 400\r",6);//set encoder
lines/rev to 400
     iresult = int(WriteCommBlock (hPol_Filter_Port,"e
```

```
400\r",6,&OL_Write_Pol));
    Sleep (1000);
//sio    iresult = SioPuts (POL_FILTER_PORT, "I 125\r",6); //set initial velocity
to 125
    iresult = int(WriteCommBlock (hPol_Filter_Port,"I
125\r",6,&OL_Write_Pol));
    Sleep (1000);
//sio    iresult = SioPuts (POL_FILTER_PORT, "V 350\r",6); //set slew velocity
to 350
    iresult = int(WriteCommBlock (hPol_Filter_Port,"V
350\r",6,&OL_Write_Pol));
    Sleep (1000);
//sio    iresult = SioPuts (POL_FILTER_PORT, "K 50 50\r",8); //set ramp
acceleration 50/50
    iresult = int(WriteCommBlock (hPol_Filter_Port,"K 50
50\r",8,&OL_Write_Pol));
    Sleep (1000);
//  find_home_filter ();
}

void CAcquire_imageDlg::find_home_filter ()
{
    int iresult;
    char s[200];
    int move_mode, nchars;
    char mess[200];

//sio    iresult = SioPuts (POL_FILTER_PORT, "F 350 1\r",8); // find home
    iresult = int(WriteCommBlock (hPol_Filter_Port,"F 350
1\r",8,&OL_Write_Pol));
    Sleep (500);
//sio    nchars = SioGets (POL_FILTER_PORT, s, 200);
    nchars = int(ReadCommBlock (hPol_Filter_Port,s,200,&OL_Read_Pol));
//  sprintf (mess, "nchars=%d", nchars);
//          iresult = MessageBox (NULL, mess, "Calibrate", MB_OKCANCEL);
//          if (iresult == IDCANCEL)
//              exit (-1);
    Sleep (500);
    move_mode = 8;
    while ((move_mode & 8) == 8) {
//sio       iresult=SioPuts (POL_FILTER_PORT, "^\r", 2);
    iresult = int(WriteCommBlock (hPol_Filter_Port,"^\r",2,&OL_Write_Pol));
        Sleep (500);
//sio       nchars = SioGets (POL_FILTER_PORT, s, 100);
    nchars = int(ReadCommBlock (hPol_Filter_Port,s,100,&OL_Read_Pol));
        s[nchars-1] = '\0';
        sscanf (&s[2], "%d", &move_mode);
//          sprintf (mess, "move status=%d", move_mode);
//          iresult = MessageBox (NULL, mess, "Calibrate", MB_OKCANCEL);
//          if (iresult == IDCANCEL)
```

```
//              exit (-1);
    }
//sio    iresult = SioPuts (POL_FILTER_PORT, "O\r", 2); // reset origin
    iresult = int(WriteCommBlock (hPol_Filter_Port,"O\r",2,&OL_Write_Pol));
    Sleep (1000);
    rotate_pol_filter (-360);
//sio    iresult = SioPuts (POL_FILTER_PORT, "O\r", 2); // reset origin
    iresult = int(WriteCommBlock (hPol_Filter_Port,"O\r",2,&OL_Write_Pol));
    Sleep (1000);
}

void CAcquire_imageDlg::rotate_pol_filter (int pos)
{
    int iresult;
    char s[100];
    int move_mode, nchars;
    int newpos;
    char mess[200];

//sio    nchars = SioGets (POL_FILTER_PORT, s, 100);
    nchars = int(ReadCommBlock (hPol_Filter_Port,s,100,&OL_Read_Pol));
    Sleep (500);
    newpos = -1;
    while (newpos != pos) {
        sprintf (s, "R+%4d\r", pos);
//sio        iresult = SioPuts (POL_FILTER_PORT, s, 7);
    iresult = int(WriteCommBlock (hPol_Filter_Port,s,7,&OL_Write_Pol));
        Sleep (1000);
//sio        nchars = SioGets (POL_FILTER_PORT, s, 100);
    nchars = int(ReadCommBlock (hPol_Filter_Port,s,100,&OL_Read_Pol));

        move_mode = 1;
        while ((move_mode % 2) > 0) {
//sio            iresult=SioPuts (POL_FILTER_PORT, "^\r", 2);
          iresult = int(WriteCommBlock
(hPol_Filter_Port,"^\r",2,&OL_Write_Pol));
            Sleep (500);
//sio            nchars = SioGets (POL_FILTER_PORT, s, 100);
        nchars = int(ReadCommBlock (hPol_Filter_Port,s,100,&OL_Read_Pol));
            s[nchars-1] = '\0';
            sscanf (&s[2], "%d", &move_mode);
        }
//sio        SioPuts (POL_FILTER_PORT, "z\r", 2);
       iresult = int(WriteCommBlock
(hPol_Filter_Port,"z\r",2,&OL_Write_Pol));
        Sleep (500);
//sio        nchars = SioGets (POL_FILTER_PORT, s, 100);
    nchars = int(ReadCommBlock (hPol_Filter_Port,s,100,&OL_Read_Pol));
        s[nchars-1] = '\0';
    }
```

```
}


//sio ClearBuffers function is not needed.
//sio   Buffers are purged with system call.
//sio  void ClearBuffers(int CN)
//sio  {
//sio    Sleep(25);
//sio    if (SioTxClear(CN)<0) SioError(SioTxClear(CN),"");
//sio    if (SioRxClear(CN)<0) SioError(SioRxClear(CN),"");
//sio    Sleep(25);
//sio  }
```

camera.cpp

```
#include "stdafx.h"
#include "camera.h"

int camera_setup ( BOOL bUseHighGain, BOOL bUseROI, int nMessageMode,
double
                exp_time, short temp)
{
  int   nResult;
  //char  szLastError[64];

  // Handle errors here
  pvSetErrorMode( PV_EM_SILENT );

  // Reset the board
  nResult = pvInitCapture( BOARDNUM );
  if ( nResult != SUCCESS )
  {
    printf("Error resetting board!\n" );
    goto fail;
  }

  // Set the device driver size expectations
  nResult = pvSetOptions( BOARDNUM, CCD_WIDTH, CCD_HEIGHT,
BITS_PER_PIXEL,
                        TIME_OUT, NUM_CHANNELS );
  if ( nResult != SUCCESS )
  {
    printf("Error setting device driver information!\n" );
    goto fail;
  }

  // Set the DLL size expectations
  nResult = pvSetCCDSize( BOARDNUM, CCD_WIDTH, CCD_HEIGHT );
  if ( nResult != SUCCESS )
  {
    printf("Error setting image size!\n" );
    goto fail;
  }

  // Set the PROM Page
  nResult = pvSetPROMPage( BOARDNUM, ( bUseHighGain ? HIGH_GAIN :
LOW_GAIN ) );
  if ( nResult != SUCCESS )
  {
    printf("Error setting PROM page!\n" );
    goto fail;
  }
```

```
  // Set the CCD Temperature.  This should happen right after Set PROM Page,
  // because setting the PROM page reset the temperature to a default value
  nResult = pvSetCCDTemperatureCalibrated( BOARDNUM, temp);
  if ( nResult != SUCCESS )
  {
    printf("Error setting CCD temperature!\n" );
    goto fail;
  }

  // Set the camera timing constants.  The Master Clock, Serial Wait, and Parallel
Wait
  //    values are of particular interest because they determine the accuracy of
  //    the exposure time.
  // If you call pvSetWaitTimes, PVAPI will perform some calculations and call
this
  //    function anyway.  It's best to call this directly if you know the values.
  nResult = pvSetWaitConstants( BOARDNUM,
                        MASTER_CLOCK,
                        DISKING_WAIT,
                        PARALLEL_WAIT,
                        AFTER_EXPO,
                        SERIAL_WAIT,
                        SKIP_WAIT );
  if ( nResult != SUCCESS )
  {
    printf("Error setting timing constants!\n" );
    goto fail;
  }

  // Set the binning
  nResult = pvSetXBinning( BOARDNUM, X_BINNING );
  if ( nResult != SUCCESS )
  {
    printf("Error setting serial binning!\n" );
    goto fail;
  }

  nResult = pvSetYBinning( BOARDNUM, Y_BINNING );
  if ( nResult != SUCCESS )
  {
    printf("Error setting parallel binning!\n" );
    goto fail;
  }

  // Set the ROI or lack thereof
  if ( bUseROI )
  {
    nResult = pvEnableSingleROI( BOARDNUM, ROI_LEFT, ROI_TOP,
ROI_RIGHT,
```

```
                                    ROI_BOTTOM );
      if ( nResult != SUCCESS )
      {
        printf("Error setting region-of-interest!\n" );
        goto fail;
      }
    }
    else
    {
      nResult = pvDisableROI( BOARDNUM );
      if ( nResult != SUCCESS )
      {
        printf("Error disabling region-of-interest!\n" );
        goto fail;
      }
    }

    // Set the exposure time
    nResult = pvSetExposureMode( BOARDNUM, PV_XM_INTERNAL,
exp_time);
    if ( nResult != SUCCESS )
    {
      printf("Error setting exposure mode!\n" );
      goto fail;
    }

    return SUCCESS;

fail:
    if ( nMessageMode != NO_MESSAGES )
    {
      if ( nMessageMode == VERBOSE_MESSAGES )
      {
        // See if the last return code tells us more
        switch ( nResult )
        {
          case ERROR_NO_DRIVER:
            printf("The VxD could not be loaded.\n" );
            break;

          case ERROR_SERIAL_INPUT_LINK_BAD:
            printf("The input link is not connected.\n" );
            break;

          case ERROR_SERIAL_LINK_BAD:
            printf("\n\nThe output serial link is not connected." );
            break;

          case ERROR_SERIAL_NO_RESPONSE:
            printf("\n\nThe camera did not respond to the serial command." );
```

```
            break;

          case ERROR_SERIAL_BAD_RESPONSE:
            printf("\n\nAn unexpected serial response was received." );
            break;

          case ERROR_SERIAL_WRITE_ERROR:
            printf("\n\nAn error occurred while writing to the serial port." );
            break;

          case ERROR_SERIAL_READ_ERROR:
            printf( "\n\nAn error occurred while reading from the serial port." );
            break;

          case ERROR_SERIAL_CANT_OPEN_PORT:
            printf( "\n\nAn error occurred while opening the serial port." );
            break;

          case ERROR_SERIAL_PORT_INIT_ERROR:
            printf( "\n\nAn error occurred while initializing the serial port." );
            break;

          default:
            break;
        }
      }

    }

    return nResult;
}
```

sercomm.cpp

```
#include "stdafx.h"
#include "sercomm.h"

//scp  begin   inserted routines

//---------------------------------------------------------------------
// int NEAR ReadCommBlock(HANDLE hSer_Comm_Port, LPSTR lpszBlock,
//                        int nMaxLength, LPOVERLAPPED lpOL_Read)
//
// Description:
//   Reads a block from the COM port and stuffs it into
//   the provided buffer.
//
// Parameters:
//   HWND hWnd
//     handle to TTY window
//
//   LPSTR lpszBlock
//     block used for storage
//
//   int nMaxLength
//     max length of block to read
//
//   LPOVERLAPPED lpOL_Read
//     pointer to structure needed for overlapped I/O
//
// Win-32 Porting Issues:
//   - ReadComm() has been replaced by ReadFile() in Win-32.
//   - Overlapped I/O has been implemented.
//
//---------------------------------------------------------------------

int NEAR ReadCommBlock(HANDLE hSer_Comm_Port, LPSTR lpszBlock,
                       int nMaxLength, LPOVERLAPPED lpOL_Read)
{
  BOOL     fReadStat ;
  COMSTAT   ComStat ;
  DWORD     dwErrorFlags;
  DWORD     dwLength;
  DWORD     dwError;
  char      szError[ 10 ] ;

  if (NULL == hSer_Comm_Port)
    return(FALSE);

  // only try to read number of bytes in queue
  ClearCommError(hSer_Comm_Port, &dwErrorFlags, &ComStat);
```

```
dwLength = min((DWORD)nMaxLength, ComStat.cbInQue);

if (dwLength > 0)
{
  fReadStat = ReadFile(hSer_Comm_Port, lpszBlock,
                       dwLength, &dwLength, lpOL_Read);
  if (!fReadStat)
  {
    if (GetLastError() == ERROR_IO_PENDING)
    {
      OutputDebugString("\n\rIO Pending");
      // We have to wait for read to complete.
      // This function will timeout according to the
      // CommTimeOuts.ReadTotalTimeoutConstant variable
      // Every time it times out, check for port errors
      while(!GetOverlappedResult(hSer_Comm_Port,
        lpOL_Read, &dwLength, TRUE))
      {
        dwError = GetLastError();
        if(dwError == ERROR_IO_INCOMPLETE)
          // normal result if not finished
          continue;
        else
        {
          // an error occurred, try to recover
          wsprintf(szError, "<CE-%u>", dwError);
          OutputDebugString(szError);
          ClearCommError(hSer_Comm_Port, &dwErrorFlags, &ComStat);
          if (dwErrorFlags > 0)
          {
              wsprintf(szError, "<CE-%u>", dwErrorFlags);
            OutputDebugString(szError);
          }
          break;
        }

      }

      }
    else
    {
      // some other error occurred
      dwLength = 0 ;
      ClearCommError(hSer_Comm_Port, &dwErrorFlags, &ComStat);
      if (dwErrorFlags > 0)
      {
          wsprintf(szError, "<CE-%u>", dwErrorFlags);
        OutputDebugString(szError);
      }
    }
```

```
    }
  }

  return(dwLength);

} // end of ReadCommBlock()

//---------------------------------------------------------------------
// BOOL NEAR WriteCommBlock(HANDLE hSer_Comm_Port, LPSTR
lpByte,
//                          DWORD dwBytesToWrite, LPOVERLAPPED
lpOL_Write)
//
// Description:
//   Writes a block of data to the Serial Comm. Port specified.
//
// Parameters:
//   HANDLE hSer_Comm_Port
//     handle to serical comm. port
//
//   LPSTR lpByte
//     pointer to data to write to port
//
//   DWORD dwBytesToWrite
//     number of bytes to write
//
//   LPOVERLAPPED lpOL_Write
//     pointer to structure needed for overlapped I/O
//
// Win-32 Porting Issues:
//   - WriteComm() has been replaced by WriteFile() in Win-32.
//   - Overlapped I/O has been implemented.
//
//---------------------------------------------------------------------

BOOL NEAR WriteCommBlock(HANDLE hSer_Comm_Port, LPSTR lpByte,
                         DWORD dwBytesToWrite, LPOVERLAPPED
lpOL_Write)
{

  BOOL       fWriteStat ;
  DWORD      dwBytesWritten ;
  DWORD      dwErrorFlags;
  DWORD      dwError;
  DWORD      dwBytesSent=0;
  COMSTAT    ComStat;
  char       szError[ 128 ] ;


  if (NULL == hSer_Comm_Port)
```

```
  return(FALSE);

fWriteStat = WriteFile(hSer_Comm_Port, lpByte, dwBytesToWrite,
                &dwBytesWritten, lpOL_Write) ;

// Note that normally the code will not execute the following
// because the driver caches write operations. Small I/O requests
// (up to several thousand bytes) will normally be accepted
// immediately and WriteFile will return true even though an
// overlapped operation was specified

if (!fWriteStat)
{
  if(GetLastError() == ERROR_IO_PENDING)
  {
    // We should wait for the completion of the write operation
    // so we know if it worked or not

    // This is only one way to do this. It might be beneficial to
    // place the write operation in a separate thread
    // so that blocking on completion will not negatively
    // affect the responsiveness of the UI

    // If the write takes too long to complete, this
    // function will timeout according to the
    // CommTimeOuts.WriteTotalTimeoutMultiplier variable.
    // This code logs the timeout but does not retry
    // the write.

    while(!GetOverlappedResult(hSer_Comm_Port,
        lpOL_Write, &dwBytesWritten, TRUE))
    {
      dwError = GetLastError();
      if(dwError == ERROR_IO_INCOMPLETE)
      {
        // normal result if not finished
        dwBytesSent += dwBytesWritten;
        continue;
      }
      else
      {
        // an error occurred, try to recover
        wsprintf( szError, "<CE-%u>", dwError);
        OutputDebugString(szError);
        ClearCommError(hSer_Comm_Port, &dwErrorFlags, &ComStat);
        if (dwErrorFlags > 0)
        {
          wsprintf(szError, "<CE-%u>", dwErrorFlags);
          OutputDebugString(szError);
        }
```

```
            break;
          }
        }

        dwBytesSent += dwBytesWritten;

        if( dwBytesSent != dwBytesToWrite )
          wsprintf(szError,"\nProbable Write Timeout: Total of %ld bytes sent",
dwBytesSent);
        else
          wsprintf(szError,"\n%ld bytes written", dwBytesSent);

        OutputDebugString(szError);

      }
      else
      {
        // some other error occurred
        ClearCommError(hSer_Comm_Port, &dwErrorFlags, &ComStat);
        if (dwErrorFlags > 0)
        {
          wsprintf(szError, "<CE-%u>", dwErrorFlags);
          OutputDebugString(szError);
        }
        return(FALSE);
      }
    }
    return(TRUE);

} // end of WriteCommBlock()

//scp  end   inserted routines
```

acquire_image.h

```
/////////////////////////////////////////////////////////////////////////////
// CAcquire_imageApp:
// See acquire_image.cpp for the implementation of this class
//

class CAcquire_imageApp : public CWinApp
{
public:
    CAcquire_imageApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAcquire_imageApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CAcquire_imageApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_ACQUIRE_IMAGE_H__ECCEE624_AEE2_11D1_81EB_0000
C0A97971__INCLUDED_)
```

acquire_imageDlg.h

```
// acquire_imageDlg.h : header file
//

#if
!defined(AFX_ACQUIRE_IMAGEDLG_H__ECCEE626_AEE2_11D1_81EB_
0000C0A97971__INCLUDED_)
#define
AFX_ACQUIRE_IMAGEDLG_H__ECCEE626_AEE2_11D1_81EB_0000C0A
97971__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "camera.h"
#include "pol_states.h"
//sio  #include "Sioerror.h"
//sio  #include "Wsc.h"

/////////////////////////////////////////////////////////////////////////////
// CAcquire_imageDlg dialog

class CAcquire_imageDlg : public CDialog
{
// Construction
public:
    CAcquire_imageDlg(CWnd* pParent = NULL); ..... // standard constructor

// Dialog Data
    //{{AFX_DATA(CAcquire_imageDlg)
    enum { IDD = IDD_ACQUIRE_IMAGE_DIALOG };
    CString   m_fname;
    int     m_max;
    int     m_min;
    int     m_mean;
    float   m_exp;
    short   m_temp;
    //}}AFX_DATA

// global stuff

    OPENFILENAME ofn2;
    char fname[500], ftitle[100];
    FILE *outfile;
    unsigned short *buffer;
//scp  serial comm port  extra type defs
   HANDLE hPol_Filter_Port;
```

```
    OVERLAPPED OL_Read_Pol,OL_Write_Pol;

    int acquire (unsigned short *);
    void OnOK ();
//scp  serial comm port  prototypes moved here for Acquire_imageDlg Class
   void pol_filter_setup();
   void rotate_pol_filter (int);
   void find_home_filter ();

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAcquire_imageDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CAcquire_imageDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnAcquireButton();
    afx_msg void OnExitButton();
    afx_msg void OnFilebrowseButton();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_ACQUIRE_IMAGEDLG_H__ECCEE626_AEE2_11D1_81EB_
0000C0A97971__INCLUDED_)
```

acq.h

```
truct stats_st {
        int   min;
        int   max;
        float mean;
};
```

camera.h

```
const unsigned short CCD_HEIGHT    = 1024;
const unsigned short BITS_PER_PIXEL = 16;
const unsigned char  BOARDNUM      = 0;
const unsigned int   TIME_OUT      = 10000;//milliseconds
const unsigned short HIGH_GAIN     = 0;   //PROM page of high gain setting
const unsigned short LOW_GAIN      = 4;   //PROM page of low gain setting
const unsigned short NUM_CHANNELS  = 1;   //number of channels (1,2 or 4)
const unsigned short CCD_TEMP      = 290; //CCD temp in Kelvin
const unsigned short X_BINNING     = 1;   //Bin this many pixels in X direction
const unsigned short Y_BINNING     = 1;   //Bin this many pixels in Y direction
const unsigned short MASTER_CLOCK  = 625; //Clock frequency is 62.5 ns
const unsigned short DISKING_WAIT  = 11;  //The camera converts this to a
time
const unsigned short PARALLEL_WAIT = 795; //"  "    "      "   " " "
const unsigned short AFTER_EXPO    = 375; //"  "    "      "   " " "
const unsigned short SERIAL_WAIT   = 2;   //"  "    "      "   " " "
const unsigned short SKIP_WAIT     = 2;   //"  "    "      "   " " "
const unsigned short ROI_LEFT      = 15; .... //use15,1038,0,1023 for full display
const unsigned short ROI_RIGHT     = 1038;
const unsigned short ROI_TOP       = 0;
const unsigned short ROI_BOTTOM    = 1023;
const unsigned short FRAME_WIDTH   = (ROI_RIGHT-ROI_LEFT+1);
const unsigned short FRAME_HEIGHT  = (ROI_BOTTOM-ROI_TOP+1);
const int MAX_INTENSITY          = 65535;
//const double EXPOSURE_TIME       = 0.03;    //exposure time in seconds
const int MINXFOV                = 435;
const int MAXXFOV                = 555;
const int MINYFOV                = 456;
const int MAXYFOV                = 574;

int camera_setup (BOOL, BOOL, int, double, short);
```

sercomm.h

```
//
// SerComm.h
//
// Use for serial communications.
//

// Comm Port definitions
#define COM1   1
#define COM2   2
#define COM3   3
#define COM4   4
#define COM5   5
#define COM6   6
#define COM7   7
#define COM8   8
#define COM9   9
#define COM10  10
#define COM11  11
#define COM12  12

// ascii XON XOFF definitions
#define ASCII_XON       0x11
#define ASCII_XOFF      0x13

// serial comm port I/O function prototypes
int NEAR ReadCommBlock(HANDLE, LPSTR, int, LPOVERLAPPED);
BOOL NEAR WriteCommBlock(HANDLE, LPSTR, DWORD,
LPOVERLAPPED);
```

Img_Bin

Files: img_bin.cpp
img_binDlg.cpp
binning.cpp
img_bin.h
img_binDlg.h

img_bin.cpp

```
// img_bin.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "img_bin.h"
#include "img_binDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CImg_binApp

BEGIN_MESSAGE_MAP(CImg_binApp, CWinApp)
    //{{AFX_MSG_MAP(CImg_binApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CImg_binApp construction

CImg_binApp::CImg_binApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CImg_binApp object
```

```
CImg_binApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CImg_binApp initialization

BOOL CImg_binApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();         ..... // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();..... // Call this when linking to MFC statically
#endif

    CImg_binDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

img_binDlg.cpp

```
// img_binDlg.cpp : implementation file
//

#include "stdafx.h"
#include "img_bin.h"
#include "img_binDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CImg_binDlg dialog

CImg_binDlg::CImg_binDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CImg_binDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CImg_binDlg)
    m_imgfile = _T("");
    m_outfile = _T("");
    m_xbin = 10;
    m_ybin = 10;
    m_calfile = _T("");
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CImg_binDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CImg_binDlg)
    DDX_Control(pDX, IDC_YBIN_SPIN, m_ybin_con);
    DDX_Control(pDX, IDC_XBIN_SPIN, m_xbin_con);
    DDX_Text(pDX, IDC_IMGFILE_EDIT, m_imgfile);
    DDX_Text(pDX, IDC_OUTFILE_EDIT, m_outfile);
    DDX_Text(pDX, IDC_XBIN_EDIT, m_xbin);
    DDX_Text(pDX, IDC_YBIN_EDIT, m_ybin);
    DDX_Text(pDX, IDC_CALFILE_BOX, m_calfile);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CImg_binDlg, CDialog)
    //{{AFX_MSG_MAP(CImg_binDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
    ON_BN_CLICKED(IDC_GO_BUTTON, OnGoButton)
```

```
    ON_BN_CLICKED(IDC_IMGBROWSE_BUTTON, OnImgbrowseButton)
    ON_BN_CLICKED(IDC_CALBROWSE_BUTTON, OnCalbrowseButton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CImg_binDlg message handlers

BOOL CImg_binDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE); // Set big icon
    SetIcon(m_hIcon, FALSE); // Set small icon

    // TODO: Add extra initialization here
    m_xbin_con.SetRange (2, 100);
    m_ybin_con.SetRange (2, 100);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CImg_binDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
```

```
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CImg_binDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CImg_binDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CImg_binDlg::OnExitButton()
{
    DestroyWindow ();
    exit (0);
}

void CImg_binDlg::OnGoButton()
```

```
{
    char fn1[100], fn2[100], fn3[100];

    UpdateData (TRUE);
    strcpy (fn1, (LPCTSTR)m_imgfile);
    if (fn1[0] == '\0') {
        MessageBox ("No Image File Selected", "img_bin", MB_OK);
        return;
    }
    strcpy (fn2, (LPCTSTR)m_outfile);
    if (fn2[0] == '\0') {
        MessageBox ("No Output File Selected", "img_bin", MB_OK);
        return;
    }
    strcpy (fn3, (LPCTSTR)m_calfile);
    if (fn3[0] == '\0') {
        MessageBox ("No Calibration File Selected", "img_bin", MB_OK);
        return;
    }

    binning (fn1, fn2, fn3, m_xbin, m_ybin);
}

// This routine gets the file name for the image file. The same
// file name with a different extension (.bct) is used for the output
// file by default.
void CImg_binDlg::OnImgbrowseButton()
{
    int iresult;
    int ext_off;
    int i;

    ofn2.lStructSize = sizeof (OPENFILENAME);
    ofn2.hInstance = NULL;
    ofn2.hwndOwner = NULL;
    ofn2.lpstrFilter = "CTISP image files (*.ctp)\0*.ctp\0All Files
(*.*)\0*.*\0\0";
    ofn2.lpstrCustomFilter = NULL;
    ofn2.nMaxCustFilter = 0;
    ofn2.nFilterIndex = 1;
    ofn2.lpstrDefExt = "ctp";
    ofn2.lCustData = NULL;
    ofn2.lpfnHook = NULL;
    ofn2.lpTemplateName = NULL;
    ofn2.lpstrFile = img_name;
    ofn2.nMaxFile = 500;
    ofn2.lpstrFileTitle = img_title;
    ofn2.nMaxFileTitle = 99;
    ofn2.lpstrInitialDir = "\\ctisp\\data";
    ofn2.lpstrTitle = "Open Image File";
```

```
    ofn2.Flags = OFN_FILEMUSTEXIST;
    img_name[0] = '\0';
    iresult = GetOpenFileName (&ofn2);
    if (iresult) {
        UpdateData (TRUE);
        m_imgfile = img_name;
        ext_off = ofn2.nFileExtension;
        for (i=0; i<ext_off; i++)
            out_name[i] = img_name[i];
        out_name[ext_off] = 'b';
        out_name[ext_off+1] = 'c';
        out_name[ext_off+2] = 't';
        out_name[ext_off+3] = '\0';
        m_outfile = out_name;
        UpdateData (FALSE);
        UpdateWindow ();
    }
}


void CImg_binDlg::OnOK ()
{
    return;
}

// This routine gets the file name for the calibration file.
void CImg_binDlg::OnCalbrowseButton()
{
    int iresult;

    ofn1.lStructSize = sizeof (OPENFILENAME);
    ofn1.hInstance = NULL;
    ofn1.hwndOwner = NULL;
    ofn1.lpstrFilter = "CTISP calibration files (*.cal)\0*.cal\0All Files
(*.*)\0*.*\0\0";
    ofn1.lpstrCustomFilter = NULL;
    ofn1.nMaxCustFilter = 0;
    ofn1.nFilterIndex = 1;
    ofn1.lpstrDefExt = "cal";
    ofn1.lCustData = NULL;
    ofn1.lpfnHook = NULL;
    ofn1.lpTemplateName = NULL;
    ofn1.lpstrFile = cal_name;
    ofn1.nMaxFile = 500;
    ofn1.lpstrFileTitle = cal_title;
    ofn1.nMaxFileTitle = 99;
    ofn1.lpstrInitialDir = "\\ctisp\\data";
    ofn1.lpstrTitle = "Open calibration File";
    ofn1.Flags = OFN_FILEMUSTEXIST;
    cal_name[0] = '\0';
```

```
    iresult = GetOpenFileName (&ofn1);
    if (iresult) {
        UpdateData (TRUE);
        m_calfile = cal_name;
        UpdateData (FALSE);
        UpdateWindow ();
    }
}
```

binning.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include "img_bin.h"
#include "img_binDlg.h"
#include <math.h>

//#define MAXX 1040
//#define MAXY 1028
#define MAXX 1024
#define MAXY 1024
#define ROI_LEFT 15
#define ROI_TOP 0

unsigned short buf1[MAXY*MAXX];
unsigned short buf2[MAXY][MAXX];

void CImg_binDlg::binning (char *infile, char *outfile, char *calfile,
                           int xbin, int ybin)
{
    FILE *in, *out, *cal;
    int i, j, k, m, kk, mm;
    int xpix, ypix, newxpix, newypix;
    int value;
    char mess[100];
    unsigned short header[13];
    int xstart, xinc, xsteps;
    int ystart, yinc, ysteps;
    int xoff, yoff;
    int nfilters;
    int nzero;
    int ixbin, iybin;

    // Open files
    //
    if ((in=fopen(infile, "rb")) == NULL) {
        MessageBox ("Error opening image file", "img_bin", MB_OK);
        return;
    }
    if ((cal=fopen(calfile, "rb")) == NULL) {
        MessageBox ("Error opening calibration file", "img_bin", MB_OK);
        return;
    }
    if ((out=fopen(outfile, "wb")) == NULL) {
        MessageBox ("Error opening output file", "img_bin", MB_OK);
        return;
    }
```

```
    // Read calibration header
    //
    fscanf (cal, "%d %d", &xpix, &ypix);
    fscanf (cal, "%d %d", &ixbin, &iybin);
    fscanf (cal, "%d %d %d", &xstart, &xinc, &xsteps);
    fscanf (cal, "%d %d %d", &ystart, &yinc, &ysteps);

    // Read image
    //
    fread (header, 2, 13, in);
    nfilters = header[0];
    xpix = header[1];
    ypix = header[2];

    // Calculate new image size
    xoff = (xstart-ROI_LEFT-(xbin/2)) % xbin;
    yoff = (ystart-ROI_TOP-(ybin/2)) % ybin + ROI_TOP;
    newxpix = (xpix-xoff) / xbin;
    newypix = (ypix-yoff) / ybin;

    // write header
    header[1] = newxpix;
    header[2] = newypix;
    fwrite (header, 2, 13, out);

    // Perform binning
    for (j=0; j<nfilters; j++) {
        fread (buf1, 2, xpix*ypix, in);
        nzero = zero (buf1, 3.0);
        for (k=0; k<newypix; k++) {
            for (m=0; m<newxpix; m++) {
                value = 0;
                for (kk=0; kk<ybin; kk++)
                    for (mm=0; mm<xbin; mm++)
                        value +=
buf1[(k*ybin+kk+yoff)*MAXX+(m*xbin+mm+xoff)];
                buf2[k][m] = value / (xbin*ybin);
            }
        }

        // Write new data
        for (i=0; i<newypix; i++)
            fwrite (&buf2[i][0], 2, newxpix, out);
    }

    MessageBox ("Done", "img_bin", MB_OK);
    fclose (in);
    fclose (cal);
    fclose (out);
```

```
}

int CImg_binDlg::zero (unsigned short *buffer, float threshold)
{
    unsigned short *buff;
    int i, j, k;
    int num;
    double sum, avg, sd;
    char mess[100];
    int val;

    sum = 0.0;
    for (i=0; i<30; i++)
        for (j=0; j<100; j++)
            sum += *(buffer+i*MAXX+j);
    avg = sum / 3000.0;
    sum = 0;
    for (i=0; i<30; i++)
        for (j=0; j<100; j++) {
            val = avg - *(buffer+i*MAXX+j);
            sum += val * val;
        }
    sd = sqrt(sum/2999.0);
    num = 0;
    buff = buffer;
    for (k=0;k<(MAXX*MAXY);k++) {
        if (((float)*buff < avg+threshold*sd))
            *buff = 0;
        else {
            *buff = *buff - avg;
            num++;
        }
        buff++;
    }
    return (num);
}
```

img_bin.h

```
// img_bin.h : main header file for the IMG_BIN application
//

#if
!defined(AFX_IMG_BIN_H__BE761B55_EA75_11D1_821E_0000C0A97971_
_INCLUDED_)
#define
AFX_IMG_BIN_H__BE761B55_EA75_11D1_821E_0000C0A97971__INCLU
DED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CImg_binApp:
// See img_bin.cpp for the implementation of this class
//

class CImg_binApp : public CWinApp
{
public:
    CImg_binApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CImg_binApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CImg_binApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
//////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_IMG_BIN_H__BE761B55_EA75_11D1_821E_0000C0A97971_
_INCLUDED_)
```

img_binDlg.h

```
// img_binDlg.h : header file
//

#if
!defined(AFX_IMG_BINDLG_H__BE761B57_EA75_11D1_821E_0000C0A97
971__INCLUDED_)
#define
AFX_IMG_BINDLG_H__BE761B57_EA75_11D1_821E_0000C0A97971__IN
CLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CImg_binDlg dialog

class CImg_binDlg : public CDialog
{
// Construction
public:
    CImg_binDlg(CWnd* pParent = NULL); ..... // standard constructor

// Dialog Data
    //{{AFX_DATA(CImg_binDlg)
    enum { IDD = IDD_IMG_BIN_DIALOG };
    CSpinButtonCtrl    m_ybin_con;
    CSpinButtonCtrl    m_xbin_con;
    CString   m_imgfile;
    CString   m_outfile;
    UINT m_xbin;
    UINT m_ybin;
    CString   m_calfile;
    //}}AFX_DATA

    // User data and functions
    char img_name[500], img_title[100];
    char out_name[500], out_title[100];
    char cal_name[500], cal_title[100];
    OPENFILENAME ofn1, ofn2, ofn3;

    void binning (char *, char *, char *, int, int);
    int zero (unsigned short *, float);

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CImg_binDlg)
    protected:
```

```
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CImg_binDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnExitButton();
    afx_msg void OnGoButton();
    afx_msg void OnImgbrowseButton();
    afx_msg void OnOK();
    afx_msg void OnCalbrowseButton();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_IMG_BINDLG_H__BE761B57_EA75_11D1_821E_0000C0A97
971__INCLUDED_)
```

Calc_Stokes

Files: reconstruction.cpp
reconstructionDlg.cpp
recon_image.cpp
matrix_inv.cpp
reconstruction.h
reconstructionDlg.h
recon.h
matrix_inv.h
StdAfx.h

reconstruction.cpp

```
// reconstruction.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "reconstruction.h"
#include "reconstructionDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CReconstructionApp

BEGIN_MESSAGE_MAP(CReconstructionApp, CWinApp)
    //{{AFX_MSG_MAP(CReconstructionApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CReconstructionApp construction

CReconstructionApp::CReconstructionApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```

```
//////////////////////////////////////////////////////////////////////////
// The one and only CReconstructionApp object

CReconstructionApp theApp;

//////////////////////////////////////////////////////////////////////////
// CReconstructionApp initialization

BOOL CReconstructionApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();      .....// Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic(); ..... // Call this when linking to MFC statically
#endif

    CReconstructionDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

reconstructionDlg.cpp

```
// reconstructionDlg.cpp : implementation file
//

#include "stdafx.h"
#include "reconstruction.h"
#include "reconstructionDlg.h"
#include "recon.h"
#include "matrix_inv.h"
#include <math.h>
#include <stdio.h>

extern struct param_st params;
extern double fmat[MAXN];
extern hcol hmat[MAXN];
extern double sum1[MAXN];

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CReconstructionDlg dialog

CReconstructionDlg::CReconstructionDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CReconstructionDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CReconstructionDlg)
    m_Status_Edit = _T("");
    m_Base_Edit = _T("");
    m_XInc_Edit = 0;
    m_XStart_Edit = 0;
    m_XSteps_Edit = 0;
    m_YInc_Edit = 0;
    m_YStart_Edit = 0;
    m_YSteps_Edit = 0;
    m_WStart_Edit = 0;
    m_WSteps_Edit = 0;
    m_WInc_Edit = 0;
    m_maxiter = 10;
    m_tol = 0.00001f;
    m_prog_text = _T("");
    m_eti = _T("");
    m_ett = _T("");
    m_est_time = _T("");
    m_outfile = _T("");
    m_imgfile = _T("");
    m_winvfile = _T("");
    m_xres = 33;
```

```
    m_yres = 33;
    m_s0_thresh = 0.001f;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CReconstructionDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CReconstructionDlg)
    DDX_Control(pDX, IDC_S0_THRESH_BOX, m_s0_con);
    DDX_Control(pDX, IDC_YRES_SPIN, m_yres_con);
    DDX_Control(pDX, IDC_YRES_BOX, m_yres_edit_con);
    DDX_Control(pDX, IDC_XRES_BOX, m_xres_edit_con);
    DDX_Control(pDX, IDC_XRES_SPIN, m_xres_con);
    DDX_Control(pDX, IDC_WINVFILE_EDIT, m_winvfile_con);
    DDX_Control(pDX, IDC_IMGFILE_EDIT, m_imgfile_con);
    DDX_Control(pDX, IDC_OUTFILE_EDIT, m_outfile_con);
    DDX_Control(pDX, IDC_TOL_EDIT, m_tol_con);
    DDX_Control(pDX, IDC_BASE_BOX, m_base_con);
    DDX_Control(pDX, IDC_MAXITER_EDIT, m_maxiter_con);
    DDX_Control(pDX, IDC_PROGRESS1, m_prog_con);
    DDX_Control(pDX, IDC_MAXITER_SPIN, m_maxiter_spin);
    DDX_Control(pDX, IDC_STATUS_BOX, m_Status_Con);
    DDX_Text(pDX, IDC_STATUS_BOX, m_Status_Edit);
    DDX_Text(pDX, IDC_BASE_BOX, m_Base_Edit);
    DDX_Text(pDX, IDC_XINC_BOX, m_XInc_Edit);
    DDX_Text(pDX, IDC_XSTART_BOX, m_XStart_Edit);
    DDX_Text(pDX, IDC_XSTEPS_BOX, m_XSteps_Edit);
    DDX_Text(pDX, IDC_YINC_BOX, m_YInc_Edit);
    DDX_Text(pDX, IDC_YSTART_BOX, m_YStart_Edit);
    DDX_Text(pDX, IDC_YSTEPS_BOX, m_YSteps_Edit);
    DDX_Text(pDX, IDC_WSTART_BOX, m_WStart_Edit);
    DDX_Text(pDX, IDC_WSTEPS_BOX, m_WSteps_Edit);
    DDX_Text(pDX, IDC_WINC_BOX, m_WInc_Edit);
    DDX_Text(pDX, IDC_MAXITER_EDIT, m_maxiter);
    DDX_Text(pDX, IDC_TOL_EDIT, m_tol);
    DDX_Text(pDX, IDC_PROG_TEXT, m_prog_text);
    DDX_Text(pDX, IDC_ETI_STATIC, m_eti);
    DDX_Text(pDX, IDC_ETT_STATIC, m_ett);
    DDX_Text(pDX, IDC_EST_STATIC, m_est_time);
    DDX_Text(pDX, IDC_OUTFILE_EDIT, m_outfile);
    DDX_Text(pDX, IDC_IMGFILE_EDIT, m_imgfile);
    DDX_Text(pDX, IDC_WINVFILE_EDIT, m_winvfile);
    DDX_Text(pDX, IDC_XRES_BOX, m_xres);
    DDX_Text(pDX, IDC_YRES_BOX, m_yres);
    DDX_Text(pDX, IDC_S0_THRESH_BOX, m_s0_thresh);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CReconstructionDlg, CDialog)
    //{{AFX_MSG_MAP(CReconstructionDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
    ON_BN_CLICKED(IDC_GO_BUTTON, OnGoButton)
    ON_BN_CLICKED(IDC_CLEARSTATUS_BUTTON,
OnClearstatusButton)
    ON_BN_CLICKED(IDC_CALBROWSE_BUTTON, OnCalbrowseButton)
    ON_BN_CLICKED(IDC_CANCEL_BUTTON, OnCancelButton)
    ON_BN_CLICKED(IDC_IMGBROWSE_BUTTON, OnImgbrowseButton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CReconstructionDlg message handlers

BOOL CReconstructionDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE)..... // Set big icon
    SetIcon(m_hIcon, FALSE);.....// Set small icon

    // TODO: Add extra initialization here

    m_xres_con.SetRange (1, 100);
```

```
    m_yres_con.SetRange (1, 100);
    m_maxiter_spin.SetRange (1, 100);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CReconstructionDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CReconstructionDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
```

```
// the minimized window.
HCURSOR CReconstructionDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CReconstructionDlg::OnExitButton()
{
    DestroyWindow ();
    exit (0);
}

void CReconstructionDlg::OnGoButton()
{
    int i, j, k, m;
    int wave;

    // Get parameters
    //
    UpdateData (TRUE);

    // Initialization
    //
    for (i=0; i<m_WSteps_Edit; i++)
        maxrec[i] = 0.0;
    params.bcancel = FALSE;
    m_maxiter_con.SetReadOnly (TRUE);
    m_xres_edit_con.SetReadOnly (TRUE);
    m_yres_edit_con.SetReadOnly (TRUE);
    m_xres_con.SetRange (m_xres, m_xres);
    m_yres_con.SetRange (m_yres, m_yres);
    m_maxiter_spin.SetRange (m_maxiter, m_maxiter);
    m_tol_con.SetReadOnly (TRUE);
    m_base_con.SetReadOnly (TRUE);
    m_outfile_con.SetReadOnly (TRUE);
    m_imgfile_con.SetReadOnly (TRUE);
    m_winvfile_con.SetReadOnly (TRUE);
    m_eti = "";
    m_ett = "";
    m_est_time = "";
    m_prog_text = "";
    m_prog_con.SetPos (0);
    m_s0_con.SetReadOnly (TRUE);

    // Open output image file
    //
    strcpy (out_fn, (LPCTSTR)m_outfile);
    if (out_fn[0] == '\0') {
        MessageBox ("No Output File Specified", "calc_stokes", MB_OK);
        return;
```

```
}
out = fopen (out_fn, "wbc");
if (out == NULL) {
    MessageBox ("Error opening output file", "calc_stokes", MB_OK);
    return;
}
m_Status_Edit += "Opened " + m_outfile + "\r\n";
UpdateData (FALSE);
update_status_scroll ();

// Open log file
//
log = fopen (log_fn, "wbc");
if (log == NULL) {
    MessageBox ("Error opening log file", "calc_stokes", MB_OK);
    return;
}
m_Status_Edit += "Opened log file \r\n";
UpdateData (FALSE);
update_status_scroll ();

// Make sure input files have been specified
//
strcpy (cal_fn, (LPCTSTR)m_Base_Edit);
if (cal_fn[0] == '\0') {
    MessageBox ("No Calibration File Specified", "calc_stokes", MB_OK);
    return;
}
OnOK ();
strcpy (inv_fn, (LPCTSTR)m_winvfile);
if (inv_fn[0] == '\0') {
    MessageBox ("No W inverse File Specified", "calc_stokes", MB_OK);
    return;
}
inv = fopen (inv_fn, "r");
if (inv == NULL) {
    MessageBox ("Error opening inverse file", "calc_stokes", MB_OK);
    return;
}
strcpy (img_fn, (LPCTSTR)m_imgfile);
if (img_fn[0] == '\0') {
    MessageBox ("No Image File Specified", "calc_stokes", MB_OK);
    return;
}


// Begin first reconstruction
//
params.rnum = 0;
time (&params.start_time);
```

```
    SendMessage (WM_START_RECON, 0, 0 ); .....
}

void CReconstructionDlg::stokes ()
{
    float winv[POL_STATES][POL_STATES];
    int i, j, k, m, n, ii, jj;
    int xsize, ysize, wsize, npol, pstate, wave;
    int ixsize, iysize, iwsize, inpol;
    float p[POL_STATES], s[7];
    float **pp, **wwinv, **ss;
    int ns, nl, num, val;
    float rval;
    unsigned char head[18];
    unsigned char *buffer;
    float *invp, *fp;
    char mess[200];
    float s2[7*MAXN], *sp;
    float r50halon[16]={14.017356,10.50370064,6.94271324,4.81366868, .....
1.56245952,1.10701032,0.95892332,0.74598888};

    sp = s2;
    // Read W inverse header
    fscanf (inv, "%d %d %d %d", &ixsize, &iysize, &iwsize, &inpol);

    // Write output header
    for (i=0; i<18; i++)
      head[i] = 0;
    ns = 7*(params.xsize+3);
    nl = params.zsize*(params.ysize+3);
    head[2] = 3;
    head[13] = ns/256;
    head[12] = ns - head[13]*256;
    head[15] = nl/256;
    head[14] = nl - head[15]*256;
    head[16] = 8;
    head[17] = 32;
    fwrite (head, 1, 18, out);

    // Allocate storage
    invp = (float
*)malloc(ixsize*iysize*POL_STATES*POL_STATES*sizeof(float));
    buffer = (unsigned char *)malloc(ns);
    ss = (float **)malloc((unsigned) POL_STATES*sizeof(float *));
    pp = (float **)malloc((unsigned) POL_STATES*sizeof(float *));
    wwinv = (float **)malloc((unsigned) POL_STATES*sizeof(float *));
    for (i=0; i<POL_STATES; i++){
        ss[i] = &s[i];
        pp[i] = &p[i];
        wwinv[i] = winv[i];
```

```
    }

    // Loop through wavelengths
    for (k=0; k<iwsize; k++) {
        // Read W inverse matrices
        fp = invp;
        for (i=0; i<ixsize*iysize; i++){
            fscanf (inv, "%d %d", &wave, &num);
            for (j=0; j<POL_STATES*POL_STATES; j++)
                fscanf (inv, "%f", fp++);
        }
        for (i=0; i<ns; i++)
            buffer[i] = 0;
        for (i=0; i<3; i++)
            fwrite (buffer, 1, ns, out);

        // Loop through xy positions
        for (i=0; i<params.ysize; i++) {
            for (j=0; j<params.xsize; j++) {
                ii = i / (params.ysize/iysize);
                jj = j / (params.xsize/ixsize);
                fp = invp + (ii*iysize+jj)*POL_STATES*POL_STATES;
                for (m=0; m<POL_STATES; m++)
                    for (n=0; n<POL_STATES; n++)
                        winv[m][n] = *(fp++);

                // Read P vector
                for (m=0; m<POL_STATES; m++)
                    p[m] =
*(rm[m]+j+i*params.xsize+k*params.xsize*params.ysize);

                // Calculate S vector and related parameters
                matrix_mult (wwinv, pp, ss, POL_STATES, POL_STATES, 1);
                if (s[0] > m_s0_thresh) {
                    s[4] = sqrt(s[1]*s[1] + s[2]*s[2] + s[3]*s[3]) / s[0];
                    s[5] = sqrt(s[1]*s[1] + s[2]*s[2]) / s[0];
                    s[6] = fabs(s[3]) / s[0];
                    s[1] = s[1] / s[0];
                    s[2] = s[2] / s[0];
                    s[3] = s[3] / s[0];
                }
                else {
                    s[1] = 0.0;
                    s[2] = 0.0;
                    s[3] = 0.0;
                    s[4] = 0.0;
                    s[5] = 0.0;
                    s[6] = 0.0;
                }
```

```
                // Normalize data for output
                for (m=0; m<7; m++) {
                    val = (s[m]+1.0) * 127.5;
                    if ((m==0) || (m>3)) val = s[m] * 255.0;
                    if (val > 255) val = 255;
                    if (val < 0) val = 0;
                    buffer[m*(params.xsize+3)+j] = (unsigned char)val;
                    *(sp++) = s[m];
                }

            }

            // Write out a line
            fwrite (buffer, 1, ns, out);
        }
    }

    // Write stokes data to log file
    //
    for (i=0; i<params.ysize; i++)
        for (j=0; j<params.xsize; j++)
            for (k=0; k<iwsize; k++) {
                fprintf (log, "%d %d %d ", j, i,
hmat[k*params.xsize*params.ysize].wave);
                for (m=0; m<7; m++)
                 fprintf (log, "%f ",
    s2[k*params.xsize*params.ysize*7+i*params.xsize*7+j*7+m]);
                fprintf (log, "\r\n");
            }

    // Write scirt file
    log2 = fopen ("temp.dat", "wbc");
    if (log2 == NULL) {
        MessageBox ("Error opening log2 file", "calc_stokes", MB_OK);
        return;
    }
    for (j=0; j<params.xsize; j++)
        for (i=0; i<params.ysize; i++)
            for (k=0; k<iwsize; k++) {
                rval = s2[k*params.xsize*params.ysize*7+i*params.xsize*7+j*7]
* 1000.0;
                if (rval > 1000.0) rval = 1000.0;
                fprintf (log2, "%f\r\n", rval);
            }
    fflush (out);
    fclose (inv);
}

void CReconstructionDlg::OnOK ()
{
```

```
    char fn[500];
    char mess[200];
    FILE *in;
    int xpix, ypix;
    int nfilters;
    int xbin, ybin;

    UpdateData (TRUE);

    // If calibration file has been selected, read it's header
    //
    strcpy (fn, (LPCTSTR)m_Base_Edit);
    if (fn[0] != '\0') {
        if ((in = fopen(fn, "r")) == NULL) {
            sprintf (mess, "Error opening %s", fn);
            MessageBox (mess, "reconstruction", MB_OK);
        }
        else {
            fscanf (in, "%d %d", &xpix, &ypix);
            fscanf (in, "%d %d", &xbin, &ybin);
            fscanf (in, "%d %d %d", &m_XStart_Edit, &m_XInc_Edit,
&m_XSteps_Edit);
            fscanf (in, "%d %d %d", &m_YStart_Edit, &m_YInc_Edit,
&m_YSteps_Edit);
            fscanf (in, "%d", &nfilters);
            fscanf (in, "%d %d %d", &m_WStart_Edit, &m_WInc_Edit,
&m_WSteps_Edit);
            UpdateData (FALSE);
            fclose (in);
        }
    }
    UpdateWindow ();
}

void CReconstructionDlg::OnClearstatusButton()
{
    UpdateData (TRUE);
    m_Status_Edit = "";
    UpdateData (FALSE);
}


void CReconstructionDlg::update_status_scroll ()
{
    int minscr, maxscr;


    m_Status_Con.GetScrollRange (SB_VERT, &minscr, &maxscr);
    if (maxscr > 11)
        m_Status_Con.LineScroll (maxscr-11, 0);
```

```
    UpdateWindow ();
}


void CReconstructionDlg::OnCalbrowseButton()
{
    int iresult;
    int ext_off;
    int i, j;
    CFileStatus status;
    char num[5]="1234";

    ofn2.lStructSize = sizeof (OPENFILENAME);
    ofn2.hInstance = NULL;
    ofn2.hwndOwner = NULL;
    ofn2.lpstrFilter = "CTISP calibration files (*.bcl)\0*.bcl\0All Files
(*.*)\0*.*\0\0";
    ofn2.lpstrCustomFilter = NULL;
    ofn2.nMaxCustFilter = 0;
    ofn2.nFilterIndex = 1;
    ofn2.lpstrDefExt = "bcl";
    ofn2.lCustData = NULL;
    ofn2.lpfnHook = NULL;
    ofn2.lpTemplateName = NULL;
    ofn2.lpstrFile = cal_name;
    ofn2.nMaxFile = 500;
    ofn2.lpstrFileTitle = cal_title;
    ofn2.nMaxFileTitle = 99;
    ofn2.lpstrInitialDir = "\\ctisp\\data";
    ofn2.lpstrTitle = "Open Calibration File";
    ofn2.Flags = OFN_FILEMUSTEXIST;
    cal_name[0] = '\0';
    iresult = GetOpenFileName (&ofn2);
    if (iresult) {
        UpdateData (TRUE);
        m_Base_Edit = cal_name;
        ext_off = ofn2.nFileExtension;
        for (i=0; i<ext_off; i++)
            winv_name[i] = cal_name[i];
        winv_name[ext_off] = 'i';
        winv_name[ext_off+1] = 'n';
        winv_name[ext_off+2] = 'v';
        winv_name[ext_off+3] = '\0';
        m_winvfile = winv_name;
        UpdateData (FALSE);
        OnOK ();
        UpdateWindow ();
        for (j=0; j<4; j++) {
            for (i=0; i<ext_off; i++)
                params.emfiles[j][i] = cal_name[i];
```

```
            params.emfiles[j][ext_off] = 'e';
            params.emfiles[j][ext_off+1] = 'm';
            params.emfiles[j][ext_off+2] = num[j];
            params.emfiles[j][ext_off+3] = '\0';
            if( CFile::GetStatus( params.emfiles[j], status ))
               params.emflags[j] = 1;
            else
               params.emflags[j] = 2;
         }
      }
}


LRESULT CReconstructionDlg::WindowProc(UINT message, WPARAM
wParam, LPARAM lParam)
{
    int i, j, k, n, m;
    char mess[200], str1[100];
    int npos;
    time_t current_time;
    int eti, ett, est;
    static int estcnt=1;
    float maxf, sf;
    unsigned char val[1024];
    float fval;
    int ind1, ind2;
    int iresult;
    char base[500];
    LPVOID param1;
    double *fp;
    int pol_order[4]={3,1,2,0};
    char *filter_name[5]={"No", "Circular", "Vertical", "45 Degree",
"Horizontal"};
    int oldwv;
    int nchars, ext_off;
    char fname[100];


    switch (message) {

       case WM_START_RECON :

          // Get cal data    for this filter and its corresponding pol state
          //
          i = params.rnum;
          sprintf (base, "Extracting cal data for %s filter\r\n",
filter_name[pol_order[i]+1]);
          m_Status_Edit += base;
          UpdateData (FALSE);
          update_status_scroll ();
```

```
            iresult = read_hcol (cal_fn, pol_order[i]+1, pol_order[i]);
            if (iresult < 0)
               return (iresult);

            // Extract image
            //
            sprintf (base, "Reading image for %s filter\r\n",
filter_name[pol_order[i]+1]);
            m_Status_Edit += base;
            UpdateData (FALSE);
            update_status_scroll ();
            iresult = read_gmat (img_fn, pol_order[i]+1);
            if (iresult < 0)
               return (iresult);

            // Perform reconstruction
            //
            sprintf (base, "Beginning reconstruction\r\n");
            m_Status_Edit += base;
            UpdateData (FALSE);
            update_status_scroll ();
            params.hwnd = m_hWnd;
            params.maxiter = m_maxiter;
            params.tolerance = m_tol;
            fprintf (log, "\r\nReconstruction for %s filter\r\n",
filter_name[pol_order[i]+1]);
            AfxBeginThread (recon_image, param1);
            return (0);

         case WM_ELEM_DONE :

            // Update progress meter
            //
            if (params.n == 0) {
               sprintf (mess, "Iteration %d", params.iter+1);
               m_prog_text = mess;
               UpdateData (FALSE);
               update_status_scroll ();
            }
            m_prog_con.SetRange (0, params.N-1);
            m_prog_con.SetPos (params.n);
            return (0);

         case WM_ITER_DONE :

            // Update time estimate
            //
            time (&current_time);
            eti = current_time - params.iter_time;
            sprintf (mess, "            Iteration Time : %02d:%02d:%02d",
```

```
            eti/3600, (eti%3600)/60, eti%60);
        m_eti = mess;
        ett = current_time - params.start_time;
        sprintf (mess, "Elapsed Time (Total) : %02d:%02d:%02d",ett/3600,
            (ett%3600)/60, ett%60);
        m_ett = mess;
        est = POL_STATES / ((float)params.rnum + (float)(params.iter+1)/
            (float)m_maxiter) * (float)ett;
        sprintf (mess, "    Estimated Time : %02d:%02d:%02d", est/3600,
            (est%3600)/60, est%60);
        m_est_time = mess;
        sprintf (str1, "Iteration %d diff = %f\r\n", params.iter+1, params.diff);
        m_Status_Edit += str1;
        UpdateData (FALSE);
        update_status_scroll ();.....
        return (0);

    case WM_RECON_DONE :

        // Write results to log file
        //
        fprintf (log, "Iteration %d diff = %f\r\n", params.iter+1, params.diff);
        for (i=0; i<params.zsize; i++) {
            fprintf (log, "wavelength = %d\r\n     ", hmat[i*params.N/
                params.zsize].wave);
            for (j=0; j<params.xsize; j++)
                fprintf (log, "%10d ", hmat[j].xpos);
            fprintf (log, "\r\n");
            for (j=0; j<params.ysize; j++) {
                fprintf (log, "y=%3d ", hmat[j*params.xsize].ypos);
                for (n=0; n<params.xsize; n++) .....
                    fprintf (log, "%10.7f ", fmat[n+j*params.xsize+
                        i*params.xsize*params.ysize]);
                fprintf (log, "\r\n");
            }
        }
        fprintf (log, "\r\n");
        fflush (log);

        // Store reconstruction results
        //
        i = params.rnum;
        rm[i] = (double *)calloc((unsigned)params.N, sizeof(double));
        fp = rm[i];
        for (m=0; m<params.N; m++) {
            *(fp+m) = fmat[m];

            j = (hmat[m].wave - m_WStart_Edit) / m_WInc_Edit;

//*************************************************************
```

```
**
                // next statement causes maximum to be calculated globally for all
wavelengths
    //*************************************************************
**
                j=0;
   //           if (*(fp+m) > maxrec[j] && hmat[m].wave == (m_WStart_Edit +
                  m_WSteps_Edit/2 *m_WInc_Edit))
//              if (*(fp+m) > maxrec[j] && hmat[m].wave > 460 &&
hmat[m].wave < 720)
                if (*(fp+m) > maxrec[j])
                    maxrec[j] = *(fp+m);
            }
            fprintf (log, "%f  Scale Factor\r\n", maxrec[0]);
            for (j=0; j<params.N; j++)
                free (hmat[j].elem);

            // Write message in status window
            //
            sprintf (str1, "Reconstruction Finished\r\n\r\n\r\n");
            m_Status_Edit += str1;
            UpdateData (FALSE);
            update_status_scroll ();

            if (params.rnum < (POL_STATES-1)) {

                // Start next reconstruction
                //
                params.rnum += 1;
                SendMessage (WM_START_RECON, 0, 0 );
            }
            else {

                // Scale reconstruction results
                //
                for (i=0; i<POL_STATES; i++) {
                    fp = rm[i];
                    for (m=0; m<params.N; m++) {
                        j = (hmat[m].wave - m_WStart_Edit) / m_WInc_Edit;
                        //
**********************************************************
                        // next statement causes scaling to be performed the same for
all wavelengths

    //*************************************************************
                        j=0;
                        *(fp+m) = (*(fp+m)) * (1.0/maxrec[j]);
                    }
                }
```

```
                // Calculate stokes vectors
                //
                sprintf (base, "Calculating stokes vectors\r\n\r\n");
                m_Status_Edit += base;
                UpdateData (FALSE);
                update_status_scroll ();
                stokes ();
                sprintf (base, "Calculation of stokes vectors for %s complete\r\n",
img_fn);
                m_Status_Edit += base;
                UpdateData (FALSE);
                update_status_scroll ();
                fclose (out);
                fclose (log);

                // If more than one file was specified, process the next one
                //
                if (file_off > 0) {
                    sprintf (fname, "%s", &img_name[file_off]);
                    m_imgfile = out_dir;
                    m_imgfile += "\\";
                    m_imgfile += fname;
                    nchars = strlen (fname);
                    ext_off = strlen(out_dir) + 1 + nchars - 3;
                    file_off = file_off + nchars + 1;
                    if (nchars > 0) {
                        strcpy (out_name, m_imgfile);
                        out_name[ext_off] = 't';
                        out_name[ext_off+1] = 'g';
                        out_name[ext_off+2] = 'a';
                        out_name[ext_off+3] = '\0';
                        m_outfile = out_name;
                        UpdateData (FALSE);
                        OnOK ();
                        UpdateWindow ();
                        strcpy (log_fn, m_imgfile);
                        log_fn[ext_off] = 'l';
                        log_fn[ext_off+1] = 'o';
                        log_fn[ext_off+2] = 'g';
                        log_fn[ext_off+3] = '\0';
                        OnGoButton ();
                        return (0);
                    }
                }

                // re-enable interface for a new calculation
                //
                MessageBox ("Done", "Calc_stokes", MB_OK);
               m_maxiter_con.SetReadOnly (FALSE);
                m_xres_edit_con.SetReadOnly (FALSE);
```

```
                m_yres_edit_con.SetReadOnly (FALSE);
                m_xres_con.SetRange (0, 100);
                m_yres_con.SetRange (0, 100);
                m_maxiter_spin.SetRange (1, 100);
                m_tol_con.SetReadOnly (FALSE);
                m_base_con.SetReadOnly (FALSE);
                m_outfile_con.SetReadOnly (FALSE);
                m_imgfile_con.SetReadOnly (FALSE);
                m_winvfile_con.SetReadOnly (FALSE);
                m_s0_con.SetReadOnly (FALSE);
            }
            return (0);
    }

    return CDialog::WindowProc(message, wParam, lParam);
}

void CReconstructionDlg::OnCancelButton()
{
    int iresult;

    iresult = MessageBox ("Cancel Reconstruction?", "Confirm", MB_YESNO);
    if (iresult == IDYES) {
        params.bcancel = TRUE;
        m_maxiter_con.SetReadOnly (FALSE);
        m_xres_edit_con.SetReadOnly (FALSE);
        m_yres_edit_con.SetReadOnly (FALSE);
        m_xres_con.SetRange (0, 100);
        m_yres_con.SetRange (0, 100);
        m_maxiter_spin.SetRange (1, 100);
        m_tol_con.SetReadOnly (FALSE);
        m_base_con.SetReadOnly (FALSE);
        m_outfile_con.SetReadOnly (FALSE);
        m_imgfile_con.SetReadOnly (FALSE);
        m_winvfile_con.SetReadOnly (FALSE);
        m_s0_con.SetReadOnly (FALSE);
        fclose (out);
        fclose (log);
    }
}



void CReconstructionDlg::OnImgbrowseButton()
{
    int iresult;
    int ext_off;
    int i;
    char fname[100];
    char dir[200];
```

```
    char mess[500];
    int nchars;

    ofn1.lStructSize = sizeof (OPENFILENAME);
    ofn1.hInstance = NULL;
    ofn1.hwndOwner = NULL;
    ofn1.lpstrFilter = "CTISP image files (*.bct)\0*.bct\0All Files
(*.*)\0*.*\0\0";
    ofn1.lpstrCustomFilter = NULL;
    ofn1.nMaxCustFilter = 0;
    ofn1.nFilterIndex = 1;
    ofn1.lpstrDefExt = "bct";
    ofn1.lCustData = NULL;
    ofn1.lpfnHook = NULL;
    ofn1.lpTemplateName = NULL;
    ofn1.lpstrFile = img_name;
    ofn1.nMaxFile = 500;
    ofn1.lpstrFileTitle = img_title;
    ofn1.nMaxFileTitle = 99;
    ofn1.lpstrInitialDir = "\\ctisp\\data";
    ofn1.lpstrTitle = "Open CTISP Image File";
    ofn1.Flags = OFN_FILEMUSTEXIST | OFN_ALLOWMULTISELECT |
OFN_EXPLORER;
    img_name[0] = '\0';
    iresult = GetOpenFileName (&ofn1);
    if (iresult) {
        ext_off = ofn1.nFileExtension;
        if (ext_off > 0){
            m_imgfile = img_name;
            file_off = 0;
        }
        else {
            file_off = ofn1.nFileOffset;
            sprintf (out_dir, "%s", img_name);
            sprintf (fname, "%s", &img_name[file_off]);
            m_imgfile = out_dir;
            m_imgfile += "\\";
            m_imgfile += fname;
            nchars = strlen (fname);
            ext_off = file_off + nchars - 3;
            file_off = file_off + nchars + 1;
        }
        strcpy (out_name, m_imgfile);
        out_name[ext_off] = 't';
        out_name[ext_off+1] = 'g';
        out_name[ext_off+2] = 'a';
        out_name[ext_off+3] = '\0';
        m_outfile = out_name;
        UpdateData (FALSE);
        OnOK ();
```

```
            UpdateWindow ();
            strcpy (log_fn, m_imgfile);
            log_fn[ext_off] = 'l';
            log_fn[ext_off+1] = 'o';
            log_fn[ext_off+2] = 'g';
            log_fn[ext_off+3] = '\0';
        }

    }
```

recon_image.cpp

```
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include "reconstruction.h"
#include "reconstructionDlg.h"
#include "recon.h"

struct param_st params;
int N;
int npol;
hcol hmat[MAXN];
hrow hmatrow[MAXPIX];
double fmat[MAXN];
double sum1[MAXN];
int *gmat;
int M;


UINT recon_image (LPVOID param1)
{
    int i, n, m, nn;
    double quot1;
    double gm, hfm, quot2, sum2;
    double newfmat[MAXN];
    arr_elem *elem, *elem2;
    int nextelem[MAXN];
    int temp_index[MAXN], temp_value[MAXN];
    int count;
    char mess[200];
    int num;
    int mx, hist[1001];
    FILE *in, *out;
    int m1;


    // Initialize F matrix
    //
    params.N = N;
    params.npol = 1;
    for (n=0; n<N; n++)
        fmat[n] = 1.0;



    if (params.emflags[params.rnum] == 1) {
```

```
        //read data
        if ((in = fopen (params.emfiles[params.rnum], "rb")) == NULL) {
            sprintf (mess, "Error opening %s", params.emfiles[params.rnum]);
            MessageBox (NULL, mess, "calc_stokes", MB_OK);
            return (-1);
        }
        fread (&m1, 4, 1, in);
        if (m1 != M) {
            sprintf (mess, "Error: m1=%d M=%d", m1, M);
            MessageBox (NULL, mess, "calc_stokes", MB_OK);
            return (-1);
        }
        for (m=0; m<M; m++) {
            fread (&hmatrow[m].num, 4, 1, in);
            if (hmatrow[m].num > 0) {
                hmatrow[m].elem = (arr_elem *)calloc(hmatrow[m].num,
sizeof(arr_elem));
                if (hmatrow[m].elem == NULL) {
                    MessageBox (NULL, "Can't allocate memory", "Recon",
MB_OK);
                    exit (-1);
                }
                fread (hmatrow[m].elem, sizeof(arr_elem), hmatrow[m].num, in);
            }
        }
        fclose (in);

    }
    else {
        // Generate H matrix stored by rows
        //
        for (n=0; n<N; n++)
            nextelem[n] = 0;
        for (m=0; m<M; m++) {
            num = 0;
            for (n=0; n<N; n++) {
                temp_index[n] = 0;
                temp_value[n] = 0;
                elem = hmat[n].elem + nextelem[n];
                if (nextelem[n] < hmat[n].num && elem->index == m) {
                    temp_index[num] = n;
                    temp_value[num] = elem->value;
                    nextelem[n]++;
                    num++;
                }
            }
            hmatrow[m].num = num;
            if (num > 0) {
                hmatrow[m].elem = (arr_elem *)calloc(num, sizeof(arr_elem));
                if (hmatrow[m].elem == NULL) {
```

```
                    MessageBox (NULL, "Can't allocate memory", "Recon",
MB_OK);
                    exit (-1);
                }
                elem = hmatrow[m].elem;
                for (i=0; i<num; i++) {
                    elem->index = temp_index[i];
                    elem->value = temp_value[i];
                    elem++;
                }
            }
        }
        // write data
        if ((out = fopen (params.emfiles[params.rnum], "wb")) == NULL) {
            sprintf (mess, "Error opening %s", params.emfiles[params.rnum]);
            MessageBox (NULL, mess, "calc_stokes", MB_OK);
            return (-1);
        }
        fwrite (&m, 4, 1, out);
        for (m=0; m<M; m++) {
            fwrite (&hmatrow[m].num, 4, 1, out);
            if (hmatrow[m].num > 0)
                fwrite (hmatrow[m].elem, sizeof(arr_elem), hmatrow[m].num,
out);
        }
        params.emflags[params.rnum] = 1;
        fclose (out);
    }


    //*****************************************************************
*********
    // Calculate Sum1
    //
    for (n=0; n<N; n++) {
        sum1[n] = 0;
        elem = hmat[n].elem;
        if (hmat[n].num > 20)
            for (m=0; m<hmat[n].num; m++) {
                sum1[n] += elem->value;
                elem++;
            }
    }

    for (i=0; i<params.maxiter; i++) {
        time (&params.iter_time);
        params.iter = i;

        // Perform an iteration
        //
```

```
    for (n=0; n<N; n++) {
        if (sum1[n] > 0)
            quot1 = fmat[n] / sum1[n];
        else
            quot1 = 0.0;

        sum2 = 0.0;
        elem = hmat[n].elem;
        for (m=0; m<hmat[n].num; m++) {
            gm = *(gmat + elem->index);
            hfm = 0.0;

            elem2 = hmatrow[elem->index].elem;
            for (nn=0; nn<hmatrow[elem->index].num; nn++) {
                hfm += elem2->value * fmat[elem2->index];
                elem2++;
            }
            if (hfm > 0)
                quot2 = gm / hfm;
            else
                quot2 = 0;
            sum2 += quot2 * elem->value;
            elem++;
        }

        newfmat[n] = quot1 * sum2;
        params.n = n;
        SendMessage (params.hwnd, WM_ELEM_DONE, 0, 0 );

        // Check for premature cancel by user
        //
        if (params.bcancel)
                AfxEndThread (-1);
    }

    // Calculate change from previous iteration
    //
    params.diff = 0.0;
    for (n=0; n<N; n++) {
        params.diff += fabs(newfmat[n]-fmat[n]);
        fmat[n] = newfmat[n];
    }
    params.diff = params.diff / N;
    SendMessage (params.hwnd, WM_ITER_DONE, 0, 0 );

    // See if difference is small enough
    //
    if (params.diff < params.tolerance) .....
        break;
}
```

```
    free (gmat);
    SendMessage (params.hwnd, WM_RECON_DONE, 0, 0 );
    return (0);
}

void getrow (int *hrow, int index)
{
    int n, m;
    arr_elem *elem;

    for (n=0; n<N; n++) {
        elem = hmat[n].elem;
        for (m=0; m<hmat[n].num; m++) {
            if (elem->index == index) {
                *(hrow+n) = elem->value;
                break;
            }
            else if (elem->index > index) {
                *(hrow+n) = 0;
                break;
            }
            elem++;
        }
    }
}

int CReconstructionDlg::read_gmat (char *filename, int filter)
{
    FILE *in;
    char mess[100];
    int i;
    int exp[NFILTERS];
    int img_order[NFILTERS];
    int xpix, ypix;
    unsigned short header[13], lbuf[1024];
    int val;
    int num;
    int j, k;

    // Open file
    //
    if ((in = fopen(filename, "rb")) == NULL) {
        sprintf (mess, "Error opening %s", filename);
        MessageBox (mess, "calc_stokes", MB_OK);
        return (-1);
    }

    // Read header
    //
    fread (header, 2, 13, in);
```

```
    num = header[0];
    if (num != NFILTERS)  {
        MessageBox ("Number of images is incorrect",
                "calc_stokes", MB_OK);
        return (-1);
    }
    xpix = header[1];
    ypix = header[2];
    if (xpix*ypix != M)     {
        MessageBox ("Size of images does not match cal file",
                "calc_stokes", MB_OK);
        return (-1);
    }
    for (i=0; i<num; i++) {
        img_order[i] = header[i+3];
        exp[i] = header[i+3+num];
//      exp[i] = 6000;
    }
    img_exp = (double)exp[filter] / 10000.0;
    sprintf (mess, "image exposure time = %f\r\n", img_exp);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    update_status_scroll ();

    // Allocate memory
    //
    gmat = (int *)calloc(M, sizeof(int));
    if (gmat == NULL) {
        sprintf (mess, "Can't allocate memory");
        MessageBox (mess, "calc_winv", MB_OK);
        return (-1);
    }

    // Read image
    fseek (in, sizeof(unsigned short)*M*img_order[filter], SEEK_CUR);
    i = 0;
    for (j=0; j<ypix; j++) {
        fread(lbuf, 2, xpix, in);
        for (k=0; k<xpix; k++) {
            val = lbuf[k] / img_exp;
            gmat[i] = val;
            i++;
        }
    }

    fclose (in);
    return (0);
}

int CReconstructionDlg:: read_hcol (char *filename, int filter, int pol)
```

```
{
    FILE *in;
    char mess[200];
    int num;
    arr_elem *elem;
    int i, j, k, l, n;
    int xpos, ypos, pstate, wv;
    int index[1000], value[1000];
    float exp_time;
    int ind;
    int xpix, ypix;
    int nfilters, iflt;
    int zz;
    int xspace, yspace;
    int kk, ll;
    int xbin, ybin;
    int l1, l2, e1, e2;
    int ii;
    float spectral_cal[16]={0.20598345, 0.38088572, 0.53272229, 0.55508733,
                            0.59775602, 0.69677914, 0.69930908, 0.72039839,
                            0.71347748, 0.72596980, 0.73453080, 0.75129487,
                            0.79075102, 0.82374279, 0.62879126, 0.50508748};

    // Open file
    //
    if ((in = fopen(filename, "r")) == NULL) {
        sprintf (mess, "Error opening %s", filename);
        MessageBox (mess, "read_hcol", MB_OK);
        return (-1);
    }

    // Read header
    //
    fscanf (in, "%d %d", &xpix, &ypix);
    fscanf (in, "%d %d", &xbin, &ybin);
    fscanf (in, "%d %d %d", &m_XStart_Edit, &m_XInc_Edit,
&m_XSteps_Edit);
    fscanf (in, "%d %d %d", &m_YStart_Edit, &m_YInc_Edit,
&m_YSteps_Edit);
    fscanf (in, "%d", &nfilters);
    fscanf (in, "%d %d %d", &m_WStart_Edit, &m_WInc_Edit,
&m_WSteps_Edit);
    fscanf (in, "%d", &npol);
    fscanf (in, "%d", &N);
    M = xpix * ypix;
    if ((m_XSteps_Edit != 1) || (m_YSteps_Edit != 1)) {
        MessageBox ("cal file has multiple positions", "Calc_stokes", MB_OK);
        return (-1);
    }
```

```
// Determine output cube parameters
//
xspace = (int)((float)FOV_SIZE / (float)m_xres + 0.5);
yspace = (int)((float)FOV_SIZE / (float)m_yres + 0.5);
params.xmin = MINXFOV + (int)(FOV_SIZE-(m_xres-1)*xspace)/2;
params.ymin = MINYFOV + (int)(FOV_SIZE-(m_yres-1)*yspace)/2;
if (m_xres == 1) params.xmin = m_XStart_Edit;
if (m_yres == 1) params.ymin = m_YStart_Edit;
params.xmax = params.xmin + (m_xres-1)*xspace;
params.ymax = params.ymin + (m_yres-1)*yspace;
params.xsize = m_xres;
params.ysize = m_yres;
params.zsize = m_WSteps_Edit;
params.xsp = xspace;
params.ysp = yspace;
// Read calibration data
//
zz = 0;
for (j=0; j<N; j++) {
    fscanf (in, "%d %d %d %d %d %f", &xpos, &ypos, &iflt, &pstate,
            &wv, &num, &exp_time);
    if ((iflt == filter) && (pstate == pol)) {
        for (i=0; i<num; i++) {
            fscanf (in, "%d %d", &index[i], &value[i]);
            value[i] = value[i] / spectral_cal[(wv-440)/20] / exp_time;
        }
        n = zz * params.xsize * params.ysize;
        for (k=0; k<m_yres; k++) {
            for (l=0; l<m_xres; l++) {
                hmat[n].num = num;
                hmat[n].xpos = params.xmin + l*xspace;
                hmat[n].ypos = params.ymin + k*yspace;
                hmat[n].pstate = pstate;
                hmat[n].wave = wv;
                hmat[n].exp_time = exp_time;
                hmat[n].elem = (arr_elem *)calloc(num, sizeof(arr_elem));
                if (hmat[n].elem == NULL) {
                    MessageBox ("Can't allocate memory", "read_hcol",
MB_OK);
                    return (-1);
                }
                if ((hmat[n].ypos-ypos) >= 0)
                    kk = (int)((float)(hmat[n].ypos-ypos) / (float)ybin + 0.5);
                else
                    kk = (int)((float)(hmat[n].ypos-ypos) / (float)ybin - 0.5);
                if ((hmat[n].xpos-xpos) >= 0)
                    ll = (int)((float)(hmat[n].xpos-xpos) / (float)xbin + 0.5);
                else
                    ll = (int)((float)(hmat[n].xpos-xpos) / (float)xbin - 0.5);
                ii =0;
```

```
                    for (i=0; i<num; i++) {
                        ind = index[i] + kk*xpix + ll;
                        l1 = index[i]/xpix;
                        l2 = ind/xpix;
                        e1 = index[i] % xpix;
                        e2 = ind % xpix;
                        if (ind >= 0 && ind < M && (l1+kk == l2)) {
                            elem = hmat[n].elem + ii;
                            elem->index = ind;
                            elem->value = value[i];
                            ii++;
                        }
                        else {
                            hmat[n].num--;
                        }
                    }
                    n++;
                }
            }
            zz++;
        }

        else
            for (i=0; i<num; i++)
                fscanf (in, "%d %d", &index, &value);
    }
    N = params.xsize * params.ysize * params.zsize;
    fclose (in);
    return (0);
}



//
// stats - calculates stats for an image
//
void stats (unsigned short *buffer, struct stats_st *image_stats)
{
    int k;
    float BufferSum;
    float avg;
    unsigned short *buff;
    int imin, imax;

    imax=*buffer;
    imin=*buffer;
    buff = buffer;
    BufferSum = 0;
```

```
    for (k=0;k<M;k++) {
        BufferSum += *buff;
        if (*buff>imax) imax=*buff;
        if ((*buff<imin) && (*buff > 0)) imin=*buff;
        buff++;
    }
    avg = BufferSum/M;
    image_stats->min = imin;
    image_stats->max = imax;
    image_stats->mean = avg;
}


// zero - zeroes all pixel values less than a threshold
//
int zero (unsigned short *buffer, float threshold)
{
    unsigned short *buff;
    int k;
    int num;

    num = 0;
    buff = buffer;
    for (k=0;k<M;k++) {
        if ((float)*buff < threshold)
            *buff = 0;
        else
            num++;
        buff++;
    }
    return (num);
}
```

matrix_inv.cpp

```
#include <math.h>
#include <stdlib.h>

#define N 4
#define TINY 1.0e-20;



void ludcmp (float **a, int n, int *indx, float *d)
{
    int i, imax, j, k;
    float big, dum, sum, temp;
    float vv[N];

    *d = 1.0;
    for (i=0; i<n; i++) {
        big = 0.0;
        for (j=0; j<n; j++)
            if ((temp=fabs(a[i][j])) > big) big = temp;
        if (big == 0.0) {
            exit (-1);
        }
        vv[i] = 1.0/big;
    }

    for (j=0; j<n; j++) {
        for (i=0; i<j; i++) {
            sum = a[i][j];
            for (k=0; k<i; k++)
                sum -= a[i][k]*a[k][j];
            a[i][j] = sum;
        }
        big = 0.0;
        for (i=j; i<n; i++) {
            sum = a[i][j];
            for (k=0; k<j; k++)
                sum -= a[i][k]*a[k][j];
            a[i][j] = sum;
            if ((dum=vv[i]*fabs(sum)) >= big) {
                big = dum;
                imax = i;
            }
        }
        if (j != imax) {
            for (k=0; k<n; k++) {
                dum = a[imax][k];
```

```
                a[imax][k] = a[j][k];
                a[j][k] = dum;
            }
            *d = -(*d);
            vv[imax] = vv[j];
        }
        indx[j] = imax;
        if (a[j][j] == 0.0)
            a[j][j] = TINY;
        if (j != (n-1)) {
            dum = 1.0/(a[j][j]);
            for (i=j+1; i<n; i++)
                a[i][j] *= dum;
        }
    }

//  free_vector (vv, 1, n);
}


void lubksb (float **a, int n, int *indx, float *b)
{
    int i, ii=-1, ip, j;
    float sum;

    for (i=0; i<n; i++) {
        ip = indx[i];
        sum = b[ip];
        b[ip] = b[i];
        if (ii >= 0)
            for (j=ii; j<=i-1; j++)
                sum -= a[i][j]*b[j];
        else if (sum)
            ii = i;
        b[i] = sum;
    }

    for (i=n-1; i>=0; i--) {
        sum = b[i];
        for (j=i+1; j<n; j++)
            sum -= a[i][j]*b[j];
        b[i] = sum/a[i][i];
    }
}



void matrix_mult (float **a, float **b, float **c, int m, int n, int r)
{
    int i, j, k;
```

```
    float sum;

    for (i=0; i<m; i++) {
        for (k=0; k<r; k++) {
            sum = 0.0;
            for (j=0; j<n; j++)
                sum += a[i][j] * b[j][k];
            c[i][k] = sum;
        }
    }
}
```

reconstruction.h

```
// reconstruction.h : main header file for the RECONSTRUCTION application
//

#if
!defined(AFX_RECONSTRUCTION_H__A1B2A085_981E_11D1_81D0_0000
C0A97971__INCLUDED_)
#define
AFX_RECONSTRUCTION_H__A1B2A085_981E_11D1_81D0_0000C0A979
71__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CReconstructionApp:
// See reconstruction.cpp for the implementation of this class
//

class CReconstructionApp : public CWinApp
{
public:
    CReconstructionApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CReconstructionApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CReconstructionApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RECONSTRUCTION_H__A1B2A085_981E_11D1_81D0_0000
C0A97971__INCLUDED_)
```

reconstructionDlg.h

```
// reconstructionDlg.h : header file
//

#if
!defined(AFX_RECONSTRUCTIONDLG_H__A1B2A087_981E_11D1_81D0_
0000C0A97971__INCLUDED_)
#define
AFX_RECONSTRUCTIONDLG_H__A1B2A087_981E_11D1_81D0_0000C0
A97971__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define POL_STATES 4
#define NFILTERS 5
// NEXT VALUES WERE 450 AND 461
#define MINXFOV 444
#define MINYFOV 458
#define FOV_SIZE 96

/////////////////////////////////////////////////////////////////////////////
// CReconstructionDlg dialog

class CReconstructionDlg : public CDialog
{
// Construction
public:
    CReconstructionDlg(CWnd* pParent = NULL); .....// standard constructor

// Dialog Data
    //{{AFX_DATA(CReconstructionDlg)
    enum { IDD = IDD_RECONSTRUCTION_DIALOG };
    CEdit m_s0_con;
    CSpinButtonCtrl    m_yres_con;
    CEdit m_yres_edit_con;
    CEdit m_xres_edit_con;
    CSpinButtonCtrl    m_xres_con;
    CEdit m_winvfile_con;
    CEdit m_imgfile_con;
    CEdit m_outfile_con;
    CEdit m_tol_con;
    CEdit m_base_con;
    CEdit m_maxiter_con;
    CProgressCtrl   m_prog_con;
    CSpinButtonCtrl    m_maxiter_spin;
    CEdit m_Status_Con;
```

```
CString   m_Status_Edit;
CString   m_Base_Edit;
int       m_XInc_Edit;
int       m_XStart_Edit;
int       m_XSteps_Edit;
int       m_YInc_Edit;
int       m_YStart_Edit;
int       m_YSteps_Edit;
int       m_WStart_Edit;
int       m_WSteps_Edit;
int       m_WInc_Edit;
int       m_maxiter;
float     m_tol;
CString   m_prog_text;
CString   m_eti;
CString   m_ett;
CString   m_est_time;
CString   m_outfile;
CString   m_imgfile;
CString   m_winvfile;
UINT m_xres;
UINT m_yres;
float   m_s0_thresh;
//}}AFX_DATA
// User defined functions and data
//
 int read_hcol (char *, int, int);
 int read_gmat (char *, int);
 void update_status_scroll ();
 void OnOK ();
 void stokes ();

 OPENFILENAME ofn1, ofn2;
 char cal_name[500], cal_title[100];
 char out_name[500];
 char img_name[500], img_title[100];
 char winv_name[500];
 FILE *out, *inv, *log, *log2;
 double *rm[POL_STATES];
 float img_exp;
 double maxrec[40];
 char out_fn[500], cal_fn[500], inv_fn[500], img_fn[500], log_fn[500];
 int file_off;
 char out_dir[200];

//
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CReconstructionDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
```

```
    virtual LRESULT WindowProc(UINT message, WPARAM wParam,
LPARAM lParam);
    //}}AFX_VIRTUAL

// Implementation

protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CReconstructionDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnExitButton();
    afx_msg void OnGoButton();
    afx_msg void OnUpdateStatusBox();
    afx_msg void OnClearstatusButton();
    afx_msg void OnCalbrowseButton();
    afx_msg void OnCancelButton();
    afx_msg void OnImgbrowseButton();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RECONSTRUCTIONDLG_H__A1B2A087_981E_11D1_81D0_
0000C0A97971__INCLUDED_)
```

recon.h

```
#define WM_ITER_DONE (WM_APP + 0) .....
#define WM_RECON_DONE (WM_APP + 1)
#define WM_ELEM_DONE (WM_APP + 2)
#define WM_START_RECON (WM_APP + 3)
#define MAXN  50000
#define MAXPIX 265000
#define IN_PROGRESS 1
#define FINISHED 2
#define ABORTED 3

struct param_st
{
    HWND   hwnd;
    int    rnum;
    int    maxiter;
    float  tolerance;
    float  diff;
    int    n;
    int    N;
    int    npol;
    int    iter;
    int    xmin;
    int    xmax;
    int    ymin;
    int    ymax;
    int    xsize;
    int    ysize;
    int    zsize;
    int    xsp;
    int    ysp;
    char   emfiles[4][100];
    int    emflags[4];
    BOOL   bcancel;
    time_t   start_time;
    time_t   iter_time;
};

struct stats_st {
        int  min;
        int  max;
        float mean;
};

typedef struct
{
    int index;
    int value;
```

```
} arr_elem;

typedef struct
{
    int num;
    int xpos;
    int ypos;
    int pstate;
    int wave;
    float exp_time;
    arr_elem *elem;
} hcol;

typedef struct
{
    int num;
    arr_elem *elem;
} hrow;


// Prototypes
//
void stats (unsigned short *, struct stats_st *);
int zero (unsigned short *, float);
void getrow (int *, int);
extern UINT recon_image (LPVOID);
extern UINT mart_recon (LPVOID);
```

matrix_inv.h

```
extern void ludcmp (float **, int, int *, float *);
extern void lubksb (float **, int, int *, float *);
extern void matrix_mult (float **, float **, float **, int, int, int);
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if
!defined(AFX_STDAFX_H__A1B2A089_981E_11D1_81D0_0000C0A97971_
_INCLUDED_)
#define
AFX_STDAFX_H__A1B2A089_981E_11D1_81D0_0000C0A97971__INCLU
DED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN... // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC OLE automation classes
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         .....// MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__A1B2A089_981E_11D1_81D0_0000C0A97971_
_INCLUDED_)
```

Cal_Bin

Files: cal_bin.cpp
cal_binDlg.cpp
binning.cpp
cal _bin.h
cal _binDlg.h

cal_bin.cpp

```
// cal_bin.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "cal_bin.h"
#include "cal_binDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCal_binApp

BEGIN_MESSAGE_MAP(CCal_binApp, CWinApp)
    //{{AFX_MSG_MAP(CCal_binApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCal_binApp construction

CCal_binApp::CCal_binApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CCal_binApp object

CCal_binApp theApp;
```

```
/////////////////////////////////////////////////////////////////////////////
// CCal_binApp initialization

BOOL CCal_binApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();    .....// Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic(); ..... // Call this when linking to MFC statically
#endif

    CCal_binDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

cal_binDlg.cpp

```
// cal_binDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cal_bin.h"
#include "cal_binDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
```

```
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCal_binDlg dialog

CCal_binDlg::CCal_binDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CCal_binDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CCal_binDlg)
    m_calfile = _T("");
    m_outfile = _T("");
    m_xbin = 10;
    m_ybin = 10;
    m_Status_Edit = _T("");
    m_thresh = 1000;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCal_binDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCal_binDlg)
    DDX_Control(pDX, IDC_STATUS_BOX, m_Status_Con);
    DDX_Control(pDX, IDC_YBIN_SPIN, m_ybin_con);
    DDX_Control(pDX, IDC_XBIN_SPIN, m_xbin_con);
    DDX_Text(pDX, IDC_CALFILE_EDIT, m_calfile);
    DDX_Text(pDX, IDC_OUTFILE_EDIT, m_outfile);
    DDX_Text(pDX, IDC_XBIN_EDIT, m_xbin);
    DDV_MinMaxUInt(pDX, m_xbin, 2, 32);
    DDX_Text(pDX, IDC_YBIN_EDIT, m_ybin);
    DDV_MinMaxUInt(pDX, m_ybin, 2, 32);
    DDX_Text(pDX, IDC_STATUS_BOX, m_Status_Edit);
    DDX_Text(pDX, IDC_THRESH_BOX, m_thresh);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCal_binDlg, CDialog)
    //{{AFX_MSG_MAP(CCal_binDlg)
    ON_WM_SYSCOMMAND()
```

```
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_CALBROWSE_BUTTON, OnCalbrowseButton)
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
    ON_BN_CLICKED(IDC_GO_BUTTON, OnGoButton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCal_binDlg message handlers

BOOL CCal_binDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);..... // Set big icon
    SetIcon(m_hIcon, FALSE);.....// Set small icon

    // Set limits for binning
    m_xbin_con.SetRange (2, 32);
    m_ybin_con.SetRange (2, 32);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CCal_binDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
```

```
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CCal_binDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CCal_binDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

// This routine gets the file name for the calibration file. The same
// file name with a different extension (.bcl) is used for the output
// file by default.
```

```
void CCal_binDlg::OnCalbrowseButton()
{
    int i, ext_off;

    UpdateData (TRUE);
    ofn2.lStructSize = sizeof (OPENFILENAME);
    ofn2.hInstance = NULL;
    ofn2.hwndOwner = NULL;
    ofn2.lpstrFilter = "Calibration files (*.cal)\0*.cal\0All Files (*.*)\0*.*\0\0";
    ofn2.lpstrCustomFilter = NULL;
    ofn2.nMaxCustFilter = 0;
    ofn2.nFilterIndex = 1;
    ofn2.lpstrDefExt = "cal";
    ofn2.lCustData = NULL;
    ofn2.lpfnHook = NULL;
    ofn2.lpTemplateName = NULL;
    ofn2.lpstrFile = cal_name;
    ofn2.nMaxFile = 500;
    ofn2.lpstrFileTitle = cal_title;
    ofn2.nMaxFileTitle = 99;
    ofn2.lpstrInitialDir = "\\ctisp\\data";
    ofn2.lpstrTitle = "Open Calibration File";
    ofn2.Flags = OFN_FILEMUSTEXIST;
    cal_name[0] = '\0';
    GetOpenFileName (&ofn2);
    m_calfile = cal_name;
    ext_off = ofn2.nFileExtension;
    for (i=0; i<ext_off; i++)
        out_name[i] = cal_name[i];
    out_name[ext_off] = 'b';
    out_name[ext_off+1] = 'c';
    out_name[ext_off+2] = 'l';
    out_name[ext_off+3] = '\0';
    m_outfile = out_name;
    UpdateData (FALSE);
    OnOK ();
    UpdateWindow ();
}

// This routine exits the program
void CCal_binDlg::OnExitButton()
{
    DestroyWindow ();
    exit (0);
}

// This routine performs binning of the specified calibration file
void CCal_binDlg::OnGoButton()
{
    char fn1[100], fn2[100];
```

```
    // Get parameters
    UpdateData (TRUE);

    // Make sure file name have been selected
    strcpy (fn1, (LPCTSTR)m_calfile);
    if (fn1[0] == '\0') {
        MessageBox ("No Calibration File Selected", "cal_bin", MB_OK);
        return;
    }
    strcpy (fn2, (LPCTSTR)m_outfile);
    if (fn2[0] == '\0') {
        MessageBox ("No Output File Selected", "cal_bin", MB_OK);
        return;
    }

    // Perform binning
    binning (fn1, fn2, m_xbin, m_ybin); .....
}

// This routine replaces the default action for the dialog. The default action,
// which is triggered by a carriage return, is to exit the dialog.
void CCal_binDlg::OnOK ()
{
    return;
}

void CCal_binDlg::update_status_scroll ()
{
    int minscr, maxscr;


    m_Status_Con.GetScrollRange (SB_VERT, &minscr, &maxscr);
    if (maxscr > 11)
        m_Status_Con.LineScroll (maxscr-11, 0);
    UpdateWindow ();
}
```

binning.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include "cal_bin.h"
#include "cal_binDlg.h"

#define MAXX 1040
#define MAXY 1028
#define MAXPTS 10000
#define ROI_LEFT 15
#define ROI_TOP 0

unsigned short buf1[MAXY][MAXX];

void CCal_binDlg::binning (char *infile, char *outfile, int xbin, int ybin)
{
    FILE *in, *out;
    int i, j, k, m, kk, mm;
    int xp, yp, wv, num;
    float exp;
    int index, value;
    int N, row, col, npol;
    int xstart, xinc, xsteps;
    int ystart, yinc, ysteps;
    int wstart, winc, wsteps;
    int xpix, ypix, newxpix, newypix;
    int pol;
    char mess[100];
    int val[MAXPTS], ind[MAXPTS];
    int xoff, yoff;
    int nfilters, iflt;
    int ixbin, iybin;

    // Open files
    //
    if ((in=fopen(infile, "r")) == NULL) {
        MessageBox ("Error opening calibration file", "cal_bin", MB_OK);
        return;
    }
    if ((out=fopen(outfile, "w")) == NULL) {
        MessageBox ("Error opening output file", "cal_bin", MB_OK);
        return;
    }

    // Read header
    //
    fscanf (in, "%d %d", &xpix, &ypix);
```

```
    if (xpix <= 0 || xpix > MAXX || ypix <= 0 || ypix > MAXY) {
        MessageBox ("Image dimensions are incorrect", "Cal_bin", MB_OK);
        return;
    }
    fscanf (in, "%d %d", &ixbin, &iybin);
    fscanf (in, "%d %d %d", &xstart, &xinc, &xsteps);
    fscanf (in, "%d %d %d", &ystart, &yinc, &ysteps);
    fscanf (in, "%d", &nfilters);
    fscanf (in, "%d %d %d", &wstart, &winc, &wsteps);
    fscanf (in, "%d", &npol);
    fscanf (in, "%d", &N);

    // Calculate new image size and write header
    xoff = (xstart-ROI_LEFT-(xbin/2)) % xbin;
    yoff = (ystart-ROI_TOP-(ybin/2)) % ybin;
    newxpix = (xpix-xoff) / xbin;
    newypix = (ypix-yoff) / ybin;
    sprintf (mess, "xoff=%d  yoff=%d  newxpix=%d  newypix=%d\r\n\r\n", xoff,
            yoff, newxpix, newypix);
    m_Status_Edit += mess;
    UpdateData (FALSE);
    fprintf (out, "%d %d\n", newxpix, newypix);
    fprintf (out, "%d %d\n", xbin, ybin);
    fprintf (out, "%d %d %d\n", xstart, xinc, xsteps);
    fprintf (out, "%d %d %d\n", ystart, yinc, ysteps);
    fprintf (out, "%d\n", nfilters);
    fprintf (out, "%d %d %d\n", wstart, winc, wsteps);
    fprintf (out, "%d\n", npol);
    fprintf (out, "%d\n", N);

    for (i=0; i<N; i++) {

        // Read data for an image
        fscanf (in,"%d %d %d %d %d %f", &xp, &yp, &iflt, &pol, &wv,
&num, &exp);
        sprintf (mess, "Input:    x=%d y=%d filter=%d pol=%d wv=%d
npts=%d\r\n",
                xp, yp, iflt, pol, wv, num);
        m_Status_Edit += mess;
        UpdateData (FALSE);
        update_status_scroll ();

        // inflate image
        for (k=0; k<MAXY; k++)
            for (m=0; m<MAXX; m++)
                buf1[k][m] = 0;
        for (j=0; j<num; j++) {
            fscanf (in, "%d %d", &index, &value);
            row = index / xpix;
            col = index % xpix;
```

```
            buf1[row][col] = value;
        }

        // Perform binning
        j=0;
        for (k=0; k<newypix; k++) {
            for (m=0; m<newxpix; m++) {
                value = 0;
                for (kk=0; kk<ybin; kk++)
                    for (mm=0; mm<xbin; mm++)
                        value += buf1[k*ybin+kk+yoff][m*xbin+mm+xoff];

                // Store only bins that are greater than threshold
                if (value > m_thresh*xbin*ybin) {
                    val[j] = value / (xbin*ybin);
                    ind[j] = k*newxpix + m;
                    j++;
                    if (j >= MAXPTS) {
                        MessageBox ("Number of nonzero bins exceeded limits",
                                "Cal_bin",    MB_OK);
                        OnExitButton();
                    }
                }
            }
        }

        // Write the binned image out
        num = j;
        fprintf (out,"%d %d %d %d %d %d %f\n", xp, yp, iflt, pol, wv, num,
exp);
        sprintf (mess, "Output: x=%d y=%d filter=%d pol=%d wv=%d
npts=%d\r\n",
                xp, yp, iflt, pol, wv, num);
        m_Status_Edit += mess;
        UpdateData (FALSE);
        update_status_scroll ();
        for (j=0; j<num; j++)
            fprintf (out, "%d %d\n", ind[j], val[j]);
    }

    MessageBox ("Done", "cal_bin", MB_OK);
}
```

cal _bin.h

```
// cal_bin.h : main header file for the CAL_BIN application
//

#if
!defined(AFX_CAL_BIN_H__6248D385_E9B1_11D1_821D_0000C0A97971_
_INCLUDED_)
#define
AFX_CAL_BIN_H__6248D385_E9B1_11D1_821D_0000C0A97971__INCLU
DED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CCal_binApp:
// See cal_bin.cpp for the implementation of this class
//

class CCal_binApp : public CWinApp
{
public:
    CCal_binApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCal_binApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CCal_binApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_CAL_BIN_H__6248D385_E9B1_11D1_821D_0000C0A97971_
_INCLUDED_)
```

cal _binDlg.h

```
// cal_binDlg.h : header file
//

#if
!defined(AFX_CAL_BINDLG_H__6248D387_E9B1_11D1_821D_0000C0A97
971__INCLUDED_)
#define
AFX_CAL_BINDLG_H__6248D387_E9B1_11D1_821D_0000C0A97971__IN
CLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CCal_binDlg dialog

class CCal_binDlg : public CDialog
{
// Construction
public:
    CCal_binDlg(CWnd* pParent = NULL);..... // standard constructor

// Dialog Data
    //{{AFX_DATA(CCal_binDlg)
    enum { IDD = IDD_CAL_BIN_DIALOG };
    CEdit m_Status_Con;
    CSpinButtonCtrl    m_ybin_con;
    CSpinButtonCtrl    m_xbin_con;
    CString   m_calfile;
    CString   m_outfile;
    UINT m_xbin;
    UINT m_ybin;
    CString   m_Status_Edit;
    UINT m_thresh;
    //}}AFX_DATA

    // User data and functions
    char cal_name[500], cal_title[100];
    char out_name[500], out_title[100];
    OPENFILENAME ofn1, ofn2;

    void binning (char *, char *, int, int);
    void update_status_scroll ();

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCal_binDlg)
```

```
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CCal_binDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnCalbrowseButton();
    afx_msg void OnExitButton();
    afx_msg void OnGoButton();
    afx_msg void OnOK ();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_CAL_BINDLG_H__6248D387_E9B1_11D1_821D_0000C0A97
971__INCLUDED_)
```

Calc_Winv

Files: reconstruction.cpp
reconstructionDlg.cpp
recon_image.cpp
matrix_inv.cpp
reconstruction.h
reconstructionDlg.h
recon.h
matrix_inv.h
StdAfx.h

reconstruction.cpp

```
// reconstruction.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "reconstruction.h"
#include "reconstructionDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CReconstructionApp

BEGIN_MESSAGE_MAP(CReconstructionApp, CWinApp)
    //{{AFX_MSG_MAP(CReconstructionApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CReconstructionApp construction

CReconstructionApp::CReconstructionApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
```

```
// The one and only CReconstructionApp object

CReconstructionApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CReconstructionApp initialization

BOOL CReconstructionApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();     ..... // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();..... // Call this when linking to MFC statically
#endif

    CReconstructionDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

reconstructionDlg.cpp

```
// reconstructionDlg.cpp : implementation file
//

#include "stdafx.h"
#include "reconstruction.h"
#include "reconstructionDlg.h"
#include "recon.h"
#include "matrix_inv.h"

extern struct param_st params;
extern double fmat[MAXN];
extern hcol hmat[MAXN];
extern double sum1[MAXN];

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CReconstructionDlg dialog

CReconstructionDlg::CReconstructionDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CReconstructionDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CReconstructionDlg)
    m_Status_Edit = _T("");
    m_Base_Edit = _T("");
    m_XInc_Edit = 0;
    m_XStart_Edit = 0;
    m_XSteps_Edit = 0;
    m_YInc_Edit = 0;
    m_YStart_Edit = 0;
    m_YSteps_Edit = 0;
    m_WStart_Edit = 0;
    m_WSteps_Edit = 0;
    m_WInc_Edit = 0;
    m_maxiter = 10;
    m_tol = 0.00001f;
    m_prog_text = _T("");
    m_eti = _T("");
    m_ett = _T("");
    m_est_time = _T("");
    m_outfile = _T("");
    m_yres = 1;
    m_xres = 1;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

```
void CReconstructionDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CReconstructionDlg)
    DDX_Control(pDX, IDC_XRES_SPIN, m_xres_con);
    DDX_Control(pDX, IDC_XRES_BOX, m_xres_edit_con);
    DDX_Control(pDX, IDC_YRES_SPIN, m_yres_con);
    DDX_Control(pDX, IDC_YRES_BOX, m_yres_edit_con);
    DDX_Control(pDX, IDC_OUTFILE_EDIT, m_outfile_con);
    DDX_Control(pDX, IDC_TOL_EDIT, m_tol_con);
    DDX_Control(pDX, IDC_BASE_BOX, m_base_con);
    DDX_Control(pDX, IDC_MAXITER_EDIT, m_maxiter_con);
    DDX_Control(pDX, IDC_PROGRESS1, m_prog_con);
    DDX_Control(pDX, IDC_MAXITER_SPIN, m_maxiter_spin);
    DDX_Control(pDX, IDC_STATUS_BOX, m_Status_Con);
    DDX_Text(pDX, IDC_STATUS_BOX, m_Status_Edit);
    DDX_Text(pDX, IDC_BASE_BOX, m_Base_Edit);
    DDX_Text(pDX, IDC_XINC_BOX, m_XInc_Edit);
    DDX_Text(pDX, IDC_XSTART_BOX, m_XStart_Edit);
    DDX_Text(pDX, IDC_XSTEPS_BOX, m_XSteps_Edit);
    DDX_Text(pDX, IDC_YINC_BOX, m_YInc_Edit);
    DDX_Text(pDX, IDC_YSTART_BOX, m_YStart_Edit);
    DDX_Text(pDX, IDC_YSTEPS_BOX, m_YSteps_Edit);
    DDX_Text(pDX, IDC_WSTART_BOX, m_WStart_Edit);
    DDX_Text(pDX, IDC_WSTEPS_BOX, m_WSteps_Edit);
    DDX_Text(pDX, IDC_WINC_BOX, m_WInc_Edit);
    DDX_Text(pDX, IDC_MAXITER_EDIT, m_maxiter);
    DDX_Text(pDX, IDC_TOL_EDIT, m_tol);
    DDX_Text(pDX, IDC_PROG_TEXT, m_prog_text);
    DDX_Text(pDX, IDC_ETI_STATIC, m_eti);
    DDX_Text(pDX, IDC_ETT_STATIC, m_ett);
    DDX_Text(pDX, IDC_EST_STATIC, m_est_time);
    DDX_Text(pDX, IDC_OUTFILE_EDIT, m_outfile);
    DDX_Text(pDX, IDC_YRES_BOX, m_yres);
    DDX_Text(pDX, IDC_XRES_BOX, m_xres);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CReconstructionDlg, CDialog)
    //{{AFX_MSG_MAP(CReconstructionDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
    ON_BN_CLICKED(IDC_GO_BUTTON, OnGoButton)
    ON_BN_CLICKED(IDC_CLEARSTATUS_BUTTON,
OnClearstatusButton)
    ON_BN_CLICKED(IDC_CALBROWSE_BUTTON, OnCalbrowseButton)
    ON_BN_CLICKED(IDC_CANCEL_BUTTON, OnCancelButton)
```

```
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CReconstructionDlg message handlers

BOOL CReconstructionDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);..... // Set big icon
    SetIcon(m_hIcon, FALSE);.....// Set small icon

    // TODO: Add extra initialization here

    m_xres_con.SetRange (0, 100);
    m_yres_con.SetRange (0, 100);
    m_maxiter_spin.SetRange (1, 100);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CReconstructionDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
```

```
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CReconstructionDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CReconstructionDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CReconstructionDlg::OnExitButton()
{
    DestroyWindow ();
    exit (0);
}

void CReconstructionDlg::OnGoButton()
```

```
{
    int iresult;
    char fn[500];
    char base[500];
    int width, height;
    LPVOID param1;
    char *ext;
    int i, j, k, m;
    int wave;
    float *fp;
    int pol_order[4]={3,1,2,0};

    // Get parameters
    //
    UpdateData (TRUE);

    // Initialization
    //
    params.bcancel = FALSE;
    m_maxiter_con.SetReadOnly (TRUE);
    m_xres_edit_con.SetReadOnly (TRUE);
    m_yres_edit_con.SetReadOnly (TRUE);
    m_xres_con.SetRange (m_xres, m_xres);
    m_yres_con.SetRange (m_yres, m_yres);
    m_maxiter_spin.SetRange (m_maxiter, m_maxiter);
    m_tol_con.SetReadOnly (TRUE);
    m_base_con.SetReadOnly (TRUE);
    m_outfile_con.SetReadOnly (TRUE);
    m_ett = "Elapsed Time (Total) : 00:00:00";
    m_est_time = "";
    m_prog_text = "";
    m_prog_con.SetPos (0);

    // Open output file
    //
    strcpy (fn, (LPCTSTR)m_outfile);
    if (fn[0] == '\0') {
        MessageBox ("No Output File Specified", "Reconstruction", MB_OK);
        return;
    }
    out = fopen (fn, "wbc");
    if (out == NULL) {
        MessageBox ("Error opening output file", "Reconstruction", MB_OK);
        return;
    }
    m_Status_Edit += "Opened " + m_outfile + "\r\n";
    UpdateData (FALSE);
    update_status_scroll ();

    // Make sure Cal file has been specified and read its header
```

```
    //
    strcpy (fn, (LPCTSTR)m_Base_Edit);
    if (fn[0] == '\0') {
        MessageBox ("No Calibration File Specified", "Reconstruction",
MB_OK);
        return;
    }
    OnOK ();

    // Write output header
    //
    fprintf (out, "%d %d %d %d\r\n", m_XSteps_Edit, m_YSteps_Edit,
                    m_WSteps_Edit, POL_STATES);

    // Loop through wavelengths
    //
    time (&params.start_time);
    for (k=0; k<m_WSteps_Edit; k++) {
        wave = m_WStart_Edit + m_WInc_Edit*k;

    // Perform reconstructions of each filter/pol state combination
    //
        for (i=0; i<NFILTERS-1; i++) {

            // Get cal data    for this filter and its corresponding pol state
            //
            sprintf (base, "Extracting cal data for filter %d \r\n", pol_order[i]+1);
            m_Status_Edit += base;
            UpdateData (FALSE);
            update_status_scroll ();
            iresult = read_hcol (fn, pol_order[i]+1, pol_order[i], wave);
            if (iresult < 0)
                return;

            for (j=0; j<POL_STATES; j++) {

                // Extract image
                //
                sprintf (base, "Extracting image for filter %d pol state %d \r\n",
                        pol_order[i]+1,pol_order[j]);
                m_Status_Edit += base;
                UpdateData (FALSE);
                update_status_scroll ();
                iresult = read_gmat (fn, pol_order[i]+1, pol_order[j], wave);
                if (iresult < 0)
                    return;

                // Perform reconstruction
                //
                recon_status = IN_PROGRESS;
```

```
                params.hwnd = m_hWnd;
                params.maxiter = m_maxiter;
                params.tolerance = m_tol;
                scale_image ();
                rm[j+i*(NFILTERS-1)] = (float *)calloc((unsigned)params.N,
sizeof(float));
                fp = rm[j+i*(NFILTERS-1)];
                for (m=0; m<params.N; m++)
                    *(fp+m) = fmat[m];
            }
            for (j=0; j<params.N; j++)
                free (hmat[j].elem);
        }
        sprintf (base, "Calculations for %dnm complete\r\n\r\n", wave);
        m_Status_Edit += base;
        UpdateData (FALSE);
        update_status_scroll ();
        W_inverse (wave);
    }
    sprintf (base, "Calculation of W inverse complete\r\n");
    m_Status_Edit += base;
    UpdateData (FALSE);
    update_status_scroll ();
    MessageBox ("Done", "Calc_winv", MB_OK);
}

void CReconstructionDlg::W_inverse (int wave)
{
    float arr[NFILTERS-1][POL_STATES]={1,1,0,0,1,-1,0,0,1,0,1,0,1,0,0,1};
    float **a, **yy, **cc, **bb, y[NFILTERS-1][POL_STATES];
    float c[NFILTERS-1][1], d, col[NFILTERS-1];
    float b[NFILTERS-1][1];
    float w[NFILTERS-1][POL_STATES], **ww, winv[NFILTERS-
1][POL_STATES];
    int i, j, k, m, indx[NFILTERS-1];
    int n;

    n = POL_STATES;
    // Allocate storage
    a = (float **)malloc((unsigned) n*sizeof(float *));
    yy = (float **)malloc((unsigned) n*sizeof(float *));
    bb = (float **)malloc((unsigned) n*sizeof(float *));
    cc = (float **)malloc((unsigned) n*sizeof(float *));
    ww = (float **)malloc((unsigned) n*sizeof(float *));
    for (i=0; i<n; i++)  {
        a[i] = arr[i];
        yy[i] = y[i];
        bb[i] = b[i];
        cc[i] = c[i];
        ww[i] = w[i];
```

```
        }
        // Invert Sm matrix
        ludcmp (a, n, indx, &d);
        for (j=0; j<n; j++) {
            for (k=0; k<n; k++)
                col[k] = 0.0;
            col[j] = 1.0;
            lubksb (a, n, indx, col);
            for (k=0; k<n; k++)
                y[k][j] = col[k];
        }
        // Loop through xy positions
        for (m=0; m<params.N; m++) {

            // Calculate W matrix
            fprintf (out, "%d %d\r\n", wave, m);
            for (j=0; j<n; j++) {
                for (k=0; k<n; k++)
                    b[k][0] = *(rm[j*n+k]+m);
                matrix_mult (yy, bb, cc, n, n, 1);
                for (k=0; k<n; k++)
                    w[j][k] = c[k][0];
            }

            // Invert W matrix
            ludcmp (ww, n, indx, &d);
            for (j=0; j<n; j++) {
                for (k=0; k<n; k++)
                    col[k] = 0.0;
                col[j] = 1.0;
                lubksb (ww, n, indx, col);
                for (k=0; k<n; k++)
                    winv[k][j] = col[k];
            }

            // Write out W inverse
            for (k=0; k<n; k++) {
                for (j=0; j<n; j++)
                    fprintf (out, "%9.5f ", winv[k][j]);
                fprintf (out, "\r\n");
            }
        }
    }

    void CReconstructionDlg::OnOK ()
    {
        char fn[500];
        char mess[200];
        FILE *in;
        int xpix, ypix;
```

```
    int nfilters;
    int xbin, ybin;

    UpdateData (TRUE);

    // If calibration file has been selected, read it's header
    //
    strcpy (fn, (LPCTSTR)m_Base_Edit);
    if (fn[0] != '\0') {
        if ((in = fopen(fn, "r")) == NULL) {
            sprintf (mess, "Error opening %s", fn);
            MessageBox (mess, "reconstruction", MB_OK);
        }
        else {
            fscanf (in, "%d %d", &xpix, &ypix);
            fscanf (in, "%d %d", &xbin, &ybin);
            fscanf (in, "%d %d %d", &m_XStart_Edit, &m_XInc_Edit,
&m_XSteps_Edit);
            fscanf (in, "%d %d %d", &m_YStart_Edit, &m_YInc_Edit,
&m_YSteps_Edit);
            fscanf (in, "%d", &nfilters);
            fscanf (in, "%d %d %d", &m_WStart_Edit, &m_WInc_Edit,
&m_WSteps_Edit);
            UpdateData (FALSE);
            fclose (in);
        }
    }
    UpdateWindow ();
}

void CReconstructionDlg::OnClearstatusButton()
{
    UpdateData (TRUE);
    m_Status_Edit = "";
    UpdateData (FALSE);
}


void CReconstructionDlg::update_status_scroll ()
{
    int minscr, maxscr;


    m_Status_Con.GetScrollRange (SB_VERT, &minscr, &maxscr);
    if (maxscr > 11)
        m_Status_Con.LineScroll (maxscr-11, 0);
    UpdateWindow ();
}

void CReconstructionDlg::OnCalbrowseButton()
```

```
{
    int iresult;
    int ext_off;
    int i;

    ofn2.lStructSize = sizeof (OPENFILENAME);
    ofn2.hInstance = NULL;
    ofn2.hwndOwner = NULL;
    ofn2.lpstrFilter = "CTISP cal files (*.bcl)\0*.bcl\0All Files (*.*)\0*.*\0\0";
    ofn2.lpstrCustomFilter = NULL;
    ofn2.nMaxCustFilter = 0;
    ofn2.nFilterIndex = 1;
    ofn2.lpstrDefExt = "bcl";
    ofn2.lCustData = NULL;
    ofn2.lpfnHook = NULL;
    ofn2.lpTemplateName = NULL;
    ofn2.lpstrFile = cal_name;
    ofn2.nMaxFile = 500;
    ofn2.lpstrFileTitle = cal_title;
    ofn2.nMaxFileTitle = 99;
    ofn2.lpstrInitialDir = "\\ctisp\\data";
    ofn2.lpstrTitle = "Open Calibration File";
    ofn2.Flags = OFN_FILEMUSTEXIST;
    cal_name[0] = '\0';
    iresult = GetOpenFileName (&ofn2);
    if (iresult) {
        UpdateData (TRUE);
        m_Base_Edit = cal_name;
        ext_off = ofn2.nFileExtension;
        for (i=0; i<ext_off; i++)
            out_name[i] = cal_name[i];
        out_name[ext_off] = 'i';
        out_name[ext_off+1] = 'n';
        out_name[ext_off+2] = 'v';
        out_name[ext_off+3] = '\0';
        m_outfile = out_name;
        UpdateData (FALSE);
        OnOK ();
        UpdateWindow ();
    }
}


LRESULT CReconstructionDlg::WindowProc(UINT message, WPARAM
wParam, LPARAM lParam)
{
    int i, j, k, n;
    char mess[200], str1[100];
    int npos;
    time_t current_time;
```

```
    int eti, ett, est;
    static int estcnt=1;
    float maxf, sf;
    unsigned char val[1024];
    float fval;
    int ind1, ind2;
    static int nrecon=0;

    switch (message) {

        case WM_ELEM_DONE :

            // Update time
            //
            return (0);

        case WM_ITER_DONE :

            // Update time estimate
            //
            return (0);

        case WM_RECON_DONE :

            // Update time

            time (&current_time);
            ett = current_time - params.start_time;
            sprintf (mess, "Elapsed Time (Total) : %02d:%02d:%02d",ett/3600,
(ett%3600)/60,
                    ett%60);
            m_ett = mess;
            UpdateData (FALSE);
            update_status_scroll ();

            // Update progress meter
            //
            nrecon++;
            m_prog_con.SetRange (0, POL_STATES*(NFILTERS-
1)*m_WSteps_Edit);
            m_prog_con.SetPos (nrecon);

            estcnt = 1;
            recon_status = FINISHED;
            return (0);
    }

    return CDialog::WindowProc(message, wParam, lParam);
}
```

```
void CReconstructionDlg::OnCancelButton()
{
    int iresult;

    iresult = MessageBox ("Cancel Reconstruction?", "Confirm", MB_YESNO);
    if (iresult == IDYES) {
        params.bcancel = TRUE;
        m_maxiter_con.SetReadOnly (FALSE);
        m_xres_edit_con.SetReadOnly (FALSE);
        m_yres_edit_con.SetReadOnly (FALSE);
        m_xres_con.SetRange (0, 100);
        m_yres_con.SetRange (0, 100);
        m_maxiter_spin.SetRange (1, 100);
        m_tol_con.SetReadOnly (FALSE);
        m_base_con.SetReadOnly (FALSE);
        recon_status = ABORTED;
    }
}
```

recon_image.cpp

```
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include "reconstruction.h"
#include "reconstructionDlg.h"
#include "recon.h"

struct param_st params;
int N;
int npol;
hcol hmat[MAXN];
hrow hmatrow[MAXPIX];
double fmat[MAXN];
double sum1[MAXN];
unsigned short *gmat;
int M;


void scale_image ()
{
    double sumh, sumg;
    arr_elem *elem;
    int m;
    char mess[200];

    // Calculate sum of H
    sumh = 0.0;
    elem = hmat[0].elem;
    if (hmat[0].num > 20)
        for (m=0; m<hmat[0].num; m++) {
            sumh += elem->value;
            elem++;
        }

    //Calculate sum of g
    for (m=0; m<M; m++)
        sumg += gmat[m];
    if (sumh > 0.0)
        fmat[0] = sumg / sumh;
    else
        fmat[0] = 0.0;
    params.N = N;
    free (gmat);
    SendMessage (params.hwnd, WM_RECON_DONE, 0, 0 );
    return;
```

```
}


void getrow (int *hrow, int index)
{
    int n, m;
    arr_elem *elem;

    for (n=0; n<N; n++) {
        *(hrow+n) = 0;
        elem = hmat[n].elem;
        for (m=0; m<hmat[n].num; m++) {
            if (elem->index == index) {
                *(hrow+n) = elem->value;
                break;
            }
            else if (elem->index > index)
                break;
            elem++;
        }
    }
}

int read_gmat (char *filename, int filter, int pol, int wave)
{
    FILE *in;
    char mess[100];
    int nfilters, iflt;
    int i, j;
    int xp, yp, wv, num;
    float exp;
    int index, value;
    int xstart, xinc, xsteps;
    int ystart, yinc, ysteps;
    int wstart, winc, wsteps;
    int xpix, ypix;
    int ipol, n;
    int xbin, ybin;

    // Open file
    //
    if ((in = fopen(filename, "r")) == NULL) {
        sprintf (mess, "Error opening %s", filename);
        MessageBox (NULL, mess, "calc_winv", MB_OK);
        return (-1);
    }

    // Read header
    //
```

```
    fscanf (in, "%d %d", &xpix, &ypix);
    fscanf (in, "%d %d", &xbin, &ybin);
    fscanf (in, "%d %d %d", &xstart, &xinc, &xsteps);
    fscanf (in, "%d %d %d", &ystart, &yinc, &ysteps);
    fscanf (in, "%d", &nfilters);
    fscanf (in, "%d %d %d", &wstart, &winc, &wsteps);
    fscanf (in, "%d", &npol);
    fscanf (in, "%d", &n);

    // Allocate memory
    //
    gmat = (unsigned short *)calloc(M, sizeof(unsigned short));
    if (gmat == NULL) {
        sprintf (mess, "Can't allocate memory");
        MessageBox (NULL, mess, "calc_winv", MB_OK);
        return (-1);
    }

    for (i=0; i<n; i++) {
        fscanf (in, "%d %d %d %d %d %d %f", &xp, &yp, &iflt, &ipol, &wv,
&num, &exp);
        if ((iflt == filter) && (ipol == pol) && (wv == wave)) {
            for (j=0; j<num; j++) {
                fscanf (in, "%d %d", &index, &value);
                if ((int)gmat[index]+value < 65535)
                    gmat[index] += value;
                else
                    gmat[index] = 65535;
            }
        }
        else
            for (j=0; j<num; j++)
                fscanf (in, "%d %d", &index, &value);
    }

    fclose (in);
    return (0);
}

int CReconstructionDlg:: read_hcol (char *filename, int filter, int pol, int wave)
{
    FILE *in;
    char mess[200];
    int num;
    arr_elem *elem;
    int i, j, k, l, n;
    int xpos, ypos, pstate, wv;
    int index[1000], value[1000];
    float exp_time;
    int ind;
```

```
    int xpix, ypix;
    int nfilters, iflt;
    int xspace, yspace;
    int kk, ll;
    int xbin, ybin;

    // Open file
    //
    if ((in = fopen(filename, "r")) == NULL) {
        sprintf (mess, "Error opening %s", filename);
        MessageBox (mess, "read_hcol", MB_OK);
        return (-1);
    }

    // Read header
    //
    fscanf (in, "%d %d", &xpix, &ypix);
    fscanf (in, "%d %d", &xbin, &ybin);
    fscanf (in, "%d %d %d", &m_XStart_Edit, &m_XInc_Edit,
&m_XSteps_Edit);
    fscanf (in, "%d %d %d", &m_YStart_Edit, &m_YInc_Edit,
&m_YSteps_Edit);
    fscanf (in, "%d", &nfilters);
    fscanf (in, "%d %d %d", &m_WStart_Edit, &m_WInc_Edit,
&m_WSteps_Edit);
    fscanf (in, "%d", &npol);
    fscanf (in, "%d", &N);
    M = xpix * ypix;
    if ((m_XSteps_Edit != 1) || (m_YSteps_Edit != 1)) {
        MessageBox ("cal file has multiple positions", "Calc_stokes", MB_OK);
        return (-1);
    }

    // Determine output cube parameters
    //
    xspace = 0;
    yspace = 0;
    params.xmin = m_XStart_Edit;
    params.xmax = m_XStart_Edit;
    params.ymin = m_YStart_Edit;
    params.ymax = m_YStart_Edit;
    params.xsize = m_xres;
    params.ysize = m_yres;
    params.zsize = 1;
    params.xsp = xspace;
    params.ysp = yspace;

    // Read calibration data
    //
    for (j=0; j<N; j++) {
```

```
        fscanf (in, "%d %d %d %d %d %d %f", &xpos, &ypos, &iflt, &pstate,
            &wv, &num, &exp_time);
//      sprintf (mess, "xpos=%d ypos=%d iflt=%d pstate=%d wv=%d num=%d",
xpos, ypos, iflt, pstate, wv, num);
//          MessageBox (mess, "Recon", MB_OK);
        if ((iflt == filter) && (pstate == pol) && (wv == wave)) {
//          sprintf (mess, "num = %d", num);
            for (i=0; i<num; i++)
                fscanf (in, "%d %d", &index[i], &value[i]);
            n = 0;
            for (k=0; k<m_yres; k++) {
                for (l=0; l<m_xres; l++) {
                    hmat[n].num = num;
                    hmat[n].xpos = params.xmin + l*xspace;
                    hmat[n].ypos = params.ymin + k*yspace;
                    hmat[n].pstate = pstate;
                    hmat[n].wave = wv;
                    hmat[n].exp_time = exp_time;
                    hmat[n].elem = (arr_elem *)calloc(num, sizeof(arr_elem));
                    if (hmat[n].elem == NULL) {
                        MessageBox ("Can't allocate memory", "read_hcol",
MB_OK);
                        return (-1);
                    }
                    kk = (int)((float)(hmat[n].ypos-ypos) / (float)ybin);
                    ll = (int)((float)(hmat[n].xpos-xpos) / (float)xbin);
                    for (i=0; i<num; i++) {
                        elem = hmat[n].elem + i;
                        ind = index[i] + kk*xpix + ll;
                        if (ind >= 0 && ind < M) {
                            elem->index = ind;
                            elem->value = value[i];
                        }
                        else {
                            elem->index = 0;
                            elem->value = 0;
                        }
                    }
                    n++;
                }
            }
        }

        else
            for (i=0; i<num; i++)
                fscanf (in, "%d %d", &index, &value);
    }
    N = params.xsize * params.ysize * params.zsize;
    fclose (in);
    return (num);
```

```
}


//
// stats - calculates stats for an image
//
void stats (unsigned short *buffer, struct stats_st *image_stats)
{
    int k;
    float BufferSum;
    float avg;
    unsigned short *buff;
    int imin, imax;

    imax=*buffer;
    imin=*buffer;
    buff = buffer;
    BufferSum = 0;
    for (k=0;k<M;k++) {
        BufferSum += *buff;
        if (*buff>imax) imax=*buff;
        if ((*buff<imin) && (*buff > 0)) imin=*buff;
        buff++;
    }
    avg = BufferSum/M;
    image_stats->min = imin;
    image_stats->max = imax;
    image_stats->mean = avg;
}


// zero - zeroes all pixel values less than a threshold
//
int zero (unsigned short *buffer, float threshold)
{
    unsigned short *buff;
    int k;
    int num;

    num = 0;
    buff = buffer;
    for (k=0;k<M;k++) {
        if ((float)*buff < threshold)
            *buff = 0;
        else
            num++;
        buff++;
    }
```

```
    return (num);
}
```

matrix_inv.cpp

```
#include "stdafx.h"
#include <math.h>
#include <stdlib.h>

#define N 4
#define TINY 1.0e-20;



void ludcmp (float **a, int n, int *indx, float *d)
{
    int i, imax, j, k;
    float big, dum, sum, temp;
    float vv[N];

    *d = 1.0;
    for (i=0; i<n; i++) {
        big = 0.0;
        for (j=0; j<n; j++)
            if ((temp=fabs(a[i][j])) > big) big = temp;
        if (big == 0.0) {
            MessageBox (NULL, "Matrix is singular", "Calc_winv", MB_OK);
            exit (-1);
        }
        vv[i] = 1.0/big;
    }

    for (j=0; j<n; j++) {
        for (i=0; i<j; i++) {
            sum = a[i][j];
            for (k=0; k<i; k++)
                sum -= a[i][k]*a[k][j];
            a[i][j] = sum;
        }
        big = 0.0;
        for (i=j; i<n; i++) {
            sum = a[i][j];
            for (k=0; k<j; k++)
                sum -= a[i][k]*a[k][j];
            a[i][j] = sum;
            if ((dum=vv[i]*fabs(sum)) >= big) {
                big = dum;
                imax = i;
            }
        }
        if (j != imax) {
```

```
            for (k=0; k<n; k++) {
                dum = a[imax][k];
                a[imax][k] = a[j][k];
                a[j][k] = dum;
            }
            *d = -(*d);
            vv[imax] = vv[j];
        }
        indx[j] = imax;
        if (a[j][j] == 0.0)
            a[j][j] = TINY;
        if (j != (n-1)) {
            dum = 1.0/(a[j][j]);
            for (i=j+1; i<n; i++)
                a[i][j] *= dum;
        }
    }

}


void lubksb (float **a, int n, int *indx, float *b)
{
    int i, ii=-1, ip, j;
    float sum;

    for (i=0; i<n; i++) {
        ip = indx[i];
        sum = b[ip];
        b[ip] = b[i];
        if (ii >= 0)
            for (j=ii; j<=i-1; j++)
                sum -= a[i][j]*b[j];
        else if (sum)
            ii = i;
        b[i] = sum;
    }

    for (i=n-1; i>=0; i--) {
        sum = b[i];
        for (j=i+1; j<n; j++)
            sum -= a[i][j]*b[j];
        b[i] = sum/a[i][i];
    }
}



void matrix_mult (float **a, float **b, float **c, int m, int n, int r)
{
```

```
    int i, j, k;
    float sum;

    for (i=0; i<m; i++) {
        for (k=0; k<r; k++) {
            sum = 0.0;
            for (j=0; j<n; j++)
                sum += a[i][j] * b[j][k];
            c[i][k] = sum;
        }
    }
}
```

reconstruction.h

```
// reconstruction.h : main header file for the RECONSTRUCTION application
//

#if
!defined(AFX_RECONSTRUCTION_H__A1B2A085_981E_11D1_81D0_0000
C0A97971__INCLUDED_)
#define
AFX_RECONSTRUCTION_H__A1B2A085_981E_11D1_81D0_0000C0A979
71__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CReconstructionApp:
// See reconstruction.cpp for the implementation of this class
//

class CReconstructionApp : public CWinApp
{
public:
    CReconstructionApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CReconstructionApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CReconstructionApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RECONSTRUCTION_H__A1B2A085_981E_11D1_81D0_0000
C0A97971__INCLUDED_)
```

reconstructionDlg.h

```
// reconstructionDlg.h : header file
//

#if
!defined(AFX_RECONSTRUCTIONDLG_H__A1B2A087_981E_11D1_81D0_
0000C0A97971__INCLUDED_)
#define
AFX_RECONSTRUCTIONDLG_H__A1B2A087_981E_11D1_81D0_0000C0
A97971__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define POL_STATES 4
#define NFILTERS 5
#define MINXFOV 435
#define MINYFOV 456
#define FOV_SIZE 120

/////////////////////////////////////////////////////////////////////////
// CReconstructionDlg dialog

class CReconstructionDlg : public CDialog
{
// Construction
public:
    CReconstructionDlg(CWnd* pParent = NULL); .....// standard constructor

// Dialog Data
    //{{AFX_DATA(CReconstructionDlg)
    enum { IDD = IDD_RECONSTRUCTION_DIALOG };
    CSpinButtonCtrl    m_xres_con;
    CEdit m_xres_edit_con;
    CSpinButtonCtrl    m_yres_con;
    CEdit m_yres_edit_con;
    CEdit m_outfile_con;
    CEdit m_tol_con;
    CEdit m_base_con;
    CEdit m_maxiter_con;
    CProgressCtrl   m_prog_con;
    CSpinButtonCtrl    m_maxiter_spin;
    CEdit m_Status_Con;
    CString   m_Status_Edit;
    CString   m_Base_Edit;
    int     m_XInc_Edit;
    int     m_XStart_Edit;
```

```
    int     m_XSteps_Edit;
    int     m_YInc_Edit;
    int     m_YStart_Edit;
    int     m_YSteps_Edit;
    int     m_WStart_Edit;
    int     m_WSteps_Edit;
    int     m_WInc_Edit;
    int     m_maxiter;
    float   m_tol;
    CString   m_prog_text;
    CString   m_eti;
    CString   m_ett;
    CString   m_est_time;
    CString   m_outfile;
    UINT m_yres;
    UINT m_xres;
    //}}AFX_DATA
    // User defined functions and data
    //
     int read_hcol (char *, int, int, int);
     void update_status_scroll ();
     void OnOK ();
     void W_inverse (int);

     OPENFILENAME ofn2, ofn3;
     char cal_name[500], cal_title[100];
     char out_name[500], out_title[100];
     FILE *out, *out2;
     int recon_status;
     float *rm[(NFILTERS-1)*POL_STATES];

    //
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CReconstructionDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    virtual LRESULT WindowProc(UINT message, WPARAM wParam,
LPARAM lParam);
    //}}AFX_VIRTUAL

// Implementation

protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CReconstructionDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
```

```
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnExitButton();
    afx_msg void OnGoButton();
    afx_msg void OnUpdateStatusBox();
    afx_msg void OnClearstatusButton();
    afx_msg void OnCalbrowseButton();
    afx_msg void OnCancelButton();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RECONSTRUCTIONDLG_H__A1B2A087_981E_11D1_81D0_
0000C0A97971__INCLUDED_)
```

recon.h

```
#define WM_ITER_DONE (WM_APP + 0) .....
#define WM_RECON_DONE (WM_APP + 1)
#define WM_ELEM_DONE (WM_APP + 2)
#define MAXN  30000
#define MAXPIX 150000
#define IN_PROGRESS 1
#define FINISHED 2
#define ABORTED 3

struct param_st
{
    HWND   hwnd;
    int    maxiter;
    float  tolerance;
    float  diff;
    int    n;
    int    N;
    int    npol;
    int    iter;
    int    xmin;
    int    xmax;
    int    ymin;
    int    ymax;
    int    xsize;
    int    ysize;
    int    zsize;
    int    xsp;
    int    ysp;
    BOOL   bcancel;
    time_t   start_time;
    time_t   iter_time;
};

struct stats_st {
        int  min;
        int  max;
        float mean;
};

typedef struct
{
    int index;
    int value;
} arr_elem;

typedef struct
{
```

```
    int num;
    int xpos;
    int ypos;
    int pstate;
    int wave;
    float exp_time;
    arr_elem *elem;
} hcol;

typedef struct
{
    int num;
    arr_elem *elem;
} hrow;

// Prototypes
//
void stats (unsigned short *, struct stats_st *);
int zero (unsigned short *, float);
void getrow (int *, int);
extern void scale_image ();
extern int read_gmat (char *, int, int, int);
```

matrix_inv.h

```
extern void ludcmp (float **, int, int *, float *);
extern void lubksb (float **, int, int *, float *);
extern void matrix_mult (float **, float **, float **, int, int, int);
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if
!defined(AFX_STDAFX_H__A1B2A089_981E_11D1_81D0_0000C0A97971_
_INCLUDED_)
#define
AFX_STDAFX_H__A1B2A089_981E_11D1_81D0_0000C0A97971__INCLU
DED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN... // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC OLE automation classes
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         .....// MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__A1B2A089_981E_11D1_81D0_0000C0A97971_
_INCLUDED_)
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
September 2001	Final report	

4. TITLE AND SUBTITLE
Computed-Tomography Imaging SpectroPolarimeter (CTISP) - A Passive Optical Sensor
Volume 1, Main Text and Appendix A; Volume 2, Appendix B

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Hollis H. (Jay) Bennett, Jr., Ricky A. Goodson, John O. Curtis

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Research and Development Center
Environmental Laboratory, In-House Laboratory Independent Research Program
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

8. PERFORMING ORGANIZATION REPORT NUMBER
ERDC/EL TR-01-32

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Assistant Secretary of the Army
Washington, DC 20315

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

An optical imaging system is described that simultaneously measures the wavelength dependence and polarization state dependence of light reflected from any surface. Potential applications of this technology include identifying man-made objects from natural backgrounds, land cover classification, and a myriad of agricultural problems such as ground moisture measurements, estimation of crop health, etc. Polarization effects are quantified through the use of Stokes parameters. Both spatial and wavelength-dependent data are collected simultaneously through the use of a phase-only computer-generated hologram as a diffraction grating. Image reconstruction is achieved through an inversion procedure called computed tomography.

15. SUBJECT TERMS
Classification system; Optical imaging system; Spectropolarimeter; Computed-tomography; Polarization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		Vol 1 - 108 Vol 2 - 211	19b. TELEPHONE NUMBER *(include area code)*
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED			

Destroy this report when no longer needed. Do not return it to the originator.